Totally Corrective Boosting Algorithms and Application to Face Recognition

Hanxi Li

June 2011

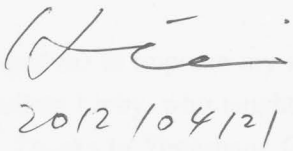
A thesis submitted for the degree of Doctor of Philosophy
of the Australian National University

For my parents and my wife.

Declaration

The work in this thesis is my own except where otherwise stated.

Hanxi Li



2012/04/21

Acknowledgements

I am profoundly grateful to my principal supervisor Chunhua Shen for his invaluable support and encouragement. It is fair to say that I wouldn't be who I am without him. Chunhua is an excellent and dedicated researcher. From him I have learnt not only research techniques, but also the devoted spirit required. I hope we can maintain this cooperation in the future. Many thanks to Hongdong Li and Lei Wang, who are my co-supervisors. I have gained lots of benefits from their great learning. I also appreciate the kind helps from Yi Li and Xumin He.

Thanks to Richard Hartley, who encouraged me to improve my academic presentation skill. Thanks to Nick Barnes and Paulette Lieby, who taught me much about doing experiments for the practical project. Thanks to Yongsheng Gao, who offered me great support and inspiration when I was writing this thesis.

I would like to express my appreciation and gratitude to my three collaborators, Qinfeng Shi, Peng Wang and Anton van den Hengel, for their excellent ideas and dedication. Qinfeng showed a solid foundation in mathematics and machine learning theory: I learned such a lot from him. Peng is a role-model for me with respect to diligence. Anton helped us all with writing English and we all appreciated his charming leadership.

I would also like to thank to my lab friends for their hard work and friendships: Junae Kim, Yongbin Zheng, Kyoungup Park, Zhihui Hao, Yiran Xie, Jun Sun, Lin Gu, Adnan Shah, Cong Phuoc Huynh ...I will never forget the days we worked together in our laboratory and every "foosball" match we played in the dining room.

Many thanks to my other friends, outside my academic life. I will not list all of you here, but my gratitude to you is immense.

Abstract

Boosting is one of the most well-known learning methods for building highly accurate classifiers or regressors from a set of weak classifiers. Much effort has been devoted to the understanding of boosting algorithms. However, questions remain unclear about the success of boosting.

In this thesis, we study boosting algorithms from a new perspective. We started our research by empirically comparing the LPBoost and AdaBoost algorithms. The result and the corresponding analysis show that, besides the minimum margin, which is directly and globally optimized in LPBoost, the margin distribution plays a more important role. Inspired by this observation, we theoretically prove that the Lagrange dual problems of AdaBoost, LogitBoost and soft-margin LPBoost with generalized hinge loss are all entropy maximization problems. By looking at the dual problems of these boosting algorithms, we show that the success of boosting algorithms can be understood in terms of maintaining a better margin distribution by maximizing margins and at the same time controlling the margin variance. We further point out that AdaBoost approximately maximizes the average margin, instead of the minimum margin. The duality formulation also enables us to develop column-generation based optimization algorithms, which are totally corrective. The new algorithm, which is termed AdaBoost-CG, exhibits almost identical classification results to those of standard stage-wise additive boosting algorithms, but with much faster convergence rates. Therefore, fewer weak classifiers are needed to build the ensemble using our proposed optimization technique.

The significance of margin distribution motivates us to design a new column-generation based algorithm that directly maximizes the average margin while minimizes the margin variance at the same time. We term this novel method MDBoost and show its superiority over other boosting-like algorithms. Moreover, consideration of the primal and dual problems together leads to important new insights into the characteristics of boosting algorithms. We then propose a general framework that can be used to design new boosting algorithms. A wide variety of machine learning problems

essentially minimize a regularized risk functional. We show that the proposed boosting framework, termed AnyBoost$_{\mathrm{TC}}$, can accommodate various loss functions and different regularizers in a totally corrective optimization way. A large body of totally corrective boosting algorithms can actually be solved very efficiently, and no sophisticated convex optimization solvers are needed, by solving the primal rather than the dual. We also demonstrate that some boosting algorithms like AdaBoost can be interpreted in our framework, even their optimization is not totally corrective, .

We conclude our study by applying the totally corrective boosting algorithm to a long-standing computer vision problem—face recognition. Linear regression face recognizers, constrained by two categories of locality, are selected and combined within both the traditional and totally corrective boosting framework. To our knowledge, it is the first time that linear-representation classifiers are boosted for face recognition. The instance-based weak classifiers bring some advantages, which are theoretically or empirically proved in our work. Benefiting from the robust weak learner and the advanced learning framework, our algorithms achieve the best reported recognition rates on face recognition benchmark datasets.

Publications

The following papers are finished based on the work presented in this thesis

1. Chunhua Shen and Hanxi Li, **On the dual formulation of boosting algorithms**, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:2216–2231, 2010.

2. Hanxi Li and Chunhua Shen, **Boosting the minimum margin: LPBoost *vs.* AdaBoost**, *International Conference on Digital Image Computing - Techniques and Applications (DICTA2008)*, Canberra, Australia, Dec. 2008.

3. Chunhua Shen and Hanxi Li, **Boosting through optimization of margin distributions**, *IEEE Transactions on Neural Networks*, 21:659–666, 2010.

4. Hanxi Li, Chunhua Shen and Yongsheng Gao, **Face Recognition Using Optimal Representation Ensembles**, submitted to *IEEE Transactions on Image Processing*, 2011.

5. Chunhua Shen, Hanxi Li and Anton van den Hengel, **Totally Corrective Boosting for Regularized Risk Minimization**, submitted to *Pattern Recognition*, 2012.

Besides the above papers, we have also been paying attention to other topics such as face recognition, compressive sensing and visual tracking. The yielded publications are listed below

1. Hanxi Li, Chunhua Shen and Qinfeng Shi, **Real-time visual tracking using compressive sensing**, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*, Colorado Springs, USA, June 2011.

2. Qinfeng Shi, Hanxi Li, and Chunhua Shen, **Rapid face recognition using hashing** [1], *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'10)*, 2753–2760, San Francisco, USA, June 2010.

[1] This paper was already presented in Dr. Shi's PhD thesis

3. Hanxi Li, Peng Wang and Chunhua Shen, **Robust face recognition via accurate face alignment and sparse representation**, *International Conference on Digital Image Computing - Techniques and Applications (DICTA2010)*, Sydney, Australia, Dec. 2010.

4. Chunhua Shen, Peng Wang, and Hanxi Li, **Lacboost and fisherboost: optimally building cascade classifiers**, *European Conference on Computer Vision (ECCV'10)*, 2:608–621, Crete Island, Greece, Sep. 2010.

5. Hanxi Li, Chunhua Shen, Qinfeng Shi and Anton van den Hengel, **Real-time Visual Tracking Using Sparse Representation**, submitted to *IEEE Transactions on Image Processing*, 2012.

6. Hanxi Li, Yongsheng Gao and Jun Sun, **Fast Kernel Sparse Representation**, submitted to *International Conference on Digital Image Computing - Techniques and Applications (DICTA2011)*, 2011.

7. Hanxi Li, Yongsheng Gao, Chunhua Shen and Lingqiao Liu, **Sparse Hough Representation for Subclass-level Object Recognition**, submitted to *European Conference on Computer Vision (ECCV'12)*, 2012.

Contents

List of Figures

List of Tables

Preliminaries, Notation, Terminology

Preliminaries

Typically, we use bold letters $\boldsymbol{u}, \boldsymbol{v}$ to denote vectors, as opposed to scalars u, v in lower case letters. We use capital letters U, V to denote matrices. All vectors are column vectors unless otherwise specified. The inner product of two column vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ is $\boldsymbol{u}^\top \boldsymbol{v} = \sum_i u_i v_i$. Component-wise inequalities are expressed using symbols $\succcurlyeq, \succ, \preccurlyeq, \prec$; *e.g.*, $\boldsymbol{u} \succcurlyeq \boldsymbol{v}$ means for all the entries $u_i \geq v_i$. $\mathbf{0}$ and $\mathbf{1}$ are column vectors with each entry being 0 and 1 respectively. The length will be clear from the context. The abbreviation s.t. means "subject to". We denote that the $\operatorname{diag}(\boldsymbol{y}) \in \mathbb{R}^{m \times m}$ is a diagonal matrix, with its (i, i) entry being label y_i, and the domain of a function $f(\cdot)$ as $\operatorname{dom} f$.

Notation for Chapter 2 to Chapter 5

$\mathbb{N}, \mathbb{R}$	natural, real numbers
$\boldsymbol{x}_i$	the ith training data
$\mathbf{y}_i$	the label corresponding to ith training data
$h(\cdot)$	a weak classifier
$F(\cdot)$	a strong classifier, *i.e.* the ensemble of several $h(\cdot)$s
$\boldsymbol{w}$	the weight vector of weak classifiers
$\boldsymbol{u}$	the weight (distribution) vector of a training set
$\mathcal{H}$	a set of weak classifiers
H	the matrix where its (i, j) entry is $H_{ij} = h_j(\boldsymbol{x}_i)$, *i.e.* the output of weak classifier h_j with respect to data $\boldsymbol{x}_i$

$H_{:j}$	the jth column of H, corresponding to the jth weak classifier
$H_{i:}$	the ith row of H, corresponding to the ith training sample
m or M	number of training examples
n or N	number of the potential weak classifiers, might be infinite
t	the iteration index of the boosting procedure
γ_i	the margin associated to the ith training sample
d	the edge of a weak classifier

Notation for Chapter 6

$\boldsymbol{x}_i$	the ith face image
N	the number of face images
N_k	the number of face images belonging to the kth individual
K	the number of individuals/classes in face recognition
$\mathbf{X}_k$	the set of faces belonging to the kth individual
$\mathbf{x}_i^k$	the ith face image for the kth individual
$\mathbf{y}$	the inquiry/test face image
r_k	the reconstruction residual of the LRC algorithm with respect to class/individual k
$l(\mathbf{y})$	the label of the test face
γ_j	the jth LRC weak classifier
$\mathcal{N}$	the neighborhood area of a given face image
ω_i	the weight of the ith face in boosting procedure
$\Psi(\mathbf{x})$	the output of the uncertain weak classifier with respect to face $\mathbf{x}$

Terminology

SVM	Support Vector Machine
LPBoost	Linear Programming Boost
LogitBoost	modified AdaBoost with logistic loss function
MadaBoost	modified AdaBoost with generalized exponential loss
CG	Column Generation
AdaBoost-QP	the boosting-like algorithm which directly optimize the margin distribution
AdaBoost-CG	totally-corrective AdaBoost using column generation
MDBoost	totally-corrective AdaBoost-QP using column generation
$\text{AnyBoost}_{\text{TC}}$	totally-corrective framework for arbitrary loss functions and regularization terms
SAMME	Stagewise Additive Modeling using a Multi-class Exponential loss function, a recently proposed multiple-class boosting algorithm
SAMME-U	SAMME with uncertain weak classifiers
SAMME-CG	totally-corrective SAMME using column geneartion
FR	Face Recognition
NFL	Nearest Feature Line algorithm
NFS	Nearest Feature Subspace algorithm
SRC	Sparse Representation Classification
LRC	Linear Regression Classification
BLRC-P	Boosted LRC based on random patches
BLRC-CG	Boosted LRC based on random patches and using SAMME-CG
BLRC-batch	Boosted LRC based on random patches and using single-step training

BLRC-PD	Boosted LRC based on random patches, with dimension-reductions for all the patches
NLRC	Neighborhood-constrained LRC
BLRC-N	Boosted LRC based on random neighborhoods

Chapter 1

Introduction

1.1 Ensemble learning

Machine learning has been playing a very important role in both scientific and industrial areas. The goal of machine learning is to give accurate predictions, usually either quantitative or categorical, based upon some existing observations [31]. As an example, consider the task of automatically identifying the risk for a certain kind of cancer. One can employ the machine learning algorithms to analyze the previously obtained clinical and demographic data and then build the associated mathematical model. After an examination is conducted on a patient, the cancer risk for him or her can be calculated based on the examining result and the mathematical model. In other words, machine learning studies how to learn a abstract model from the real-life observations and how to use this model to infer the unknown properties of the new samples.

In practice, given a set of observations, usually one can obtain a number of basic predicting models, which may generate different predictions for the same sample. Intuitively, to combine the basic predictions in some way can lead to a better result, and the combining method governs the final performance. In the literature of machine learning, the methods that build a (better) predicting model by combining the strengths of some basic models are categorized as ensemble learning algorithms.

There are several well-known ensemble learning algorithms. For example, the **B**ootstrap **agg**regat**ing** (Bagging) algorithm [5] combines the tree classifiers learned from randomly-selected subsets of the original training data. The final prediction of the Bagging algorithm is simply the average value of all the basic predictions. As an enhanced version of Bagging, Random Forests improve the prediction's accuracy by reducing the correlation between the tree classifiers [9].

Boosting is another kind of ensemble learning algorithm. Differing from the con-

ventional voting strategy employed by Bagging and random forests, boosting conducts a weighted voting procedure over all the basic models. The boosting algorithm was originally proposed for solving classification problems, where the model predictions are categorical and usually indicate the samples' identities. Typically, in boosting algorithms, each basic model (usually referred to as base classifier or weak classifiers in the literature) * generated by a weak learning algorithm (usually called weak learner) has a misclassification error that is slightly better than a random guess. Then the weak classifiers are assigned with different weights which stand for the different significances in the voting procedure. The weighted combination of the weak classifier, usually referred to as the strong classifier, has a much lower test error. In this sense, boosting algorithms can "boost" the weak learning algorithm to obtain a much stronger classifier. With the properly learned weights, boosting algorithms usually illustrate higher accuracies than Bagging.

1.2 AdaBoost

Boosting has attracted a lot of research interests since the first practical boosting algorithm, AdaBoost (**Ada**ptive **Boost**ing), was introduced by Freund and Schapire [28]. As indicated by its name, AdaBoost adjusts itself adaptively to the errors of the weak classifiers [28]. In specific, AdaBoost trains the classifier in a iterative style. At each step, one weak classifier is learned by using the weak learner, *e.g.* decision trees or Principle Component Analysis (PCA), over a weighted data distribution. After obtained the current weak classifier, the data distribution is updated so that higher weights are given to the misclassified samples. In this way, the newly added weak classifier will always focus on the "hard samples" and the combined strong classifier usually has much higher discriminative power. The AdaBoost algorithm was proposed for effectively reducing the training errors [73]. Nonetheless, it also demonstrates low test errors in most scenarios. Actually, AdaBoost with trees was regarded as the "best off-the-shelf classifier in the world" [7].

Another interesting property of AdaBoost is the high resistance to overfittings. For most machine learning algorithms, if there is no constraints on model's complexity, usually the learned model will "overfit" the training data. That is to say, the model scarifies the generalization capability for the low training error. In contrast, the test error of AdaBoost seems to consistently decrease when more and more weak classifiers are added into the model.

*We will use the words *base classifier* and *weak classifier* interchangeably.

1.3 The interpretations and variations of AdaBoost

After AdaBoost was proposed, the machine learning community has demonstrated great interests in its desirable properties. Much work has been done to analyze AdaBoost and to interpret its success. The first generalization error bound of AdaBoost was introduced in the same paper where the AdaBoost algorithm was proposed [28]. The bound is determined by the training error, the size of the training set, the VC-dimension [89] of the base classifier space and the number of the boosting rounds. However, it can not explain AdaBoost's resistance to overfitting, as many early experiments [6, 60] suggested. Later, inspired by the theoretical analysis made by Bartlett [1], Schapire *et al.* introduced a new generalization error bound for AdaBoost. This bound is entirely independent of the boosting round number and shows an explicit link to the margin distribution of training samples. They also proved that AdaBoost is able to achieve a good margin distribution. However, Breiman [8] cast serious doubts on this explanation. He designed a boosting-like algorithms called Arc-Gv, which usually generates larger minimum margins than AdaBoost. A sharper upper bound of generalization error was built in terms of minimum margin, the training set size and the number of base classifiers. Breiman assumed that his Arc-Gv would achieve higher performance than AdaBoost according to the new bound, while the experimental result didn't support this assumption. During the past decade, more sophisticated margin-based explanations have been proposed [64, 71, 92].

From the perspective of optimization, also much research has been dedicated to interpret AdaBoost. The behavior of AdaBoost was originally understood in a game-theoretic setting as explored by Freund and Schapire [28]. The zero-sum game is well-known to be solvable using linear programming. Therefore AdaBoost has been linked to linear or more general convex programming procedures [35, 63, 16]. Friedman *et al.* [29] theoretically established the strong connection between AdaBoost and logistic regression. In addition, Breiman [8] and Friedman *et al.* [30] observed that many boosting algorithms can be viewed as gradient descent optimization in functional space. Rosset and his colleagues proved that some boosting algorithms can be viewed as ℓ_1-norm regularized model fitting [68].

According to the different interpretations, different variations of AdaBoost are also designed. Arc-Gv was derived to maximize the minimum margin based on Breiman's explanation [8]; Friedman *et al.* [29] proposed LogitBoost by replacing AdaBoost's exponential cost function with the loss function of logistic regression. MadaBoost [20] instead uses a modified exponential loss. Motivated by the success of the margin theory associated with support vector machines (SVMs), Demiriz *et al.* [35, 16] invented LP-

Boost with the intuition of maximizing the minimum margin of all training examples. ε-Boosting algorithm was developed by introducing the connection between AdaBoost and the barrier optimization [63]. The coordinate ascent boosting algorithms [70, 71] was proposed based on the analysis of the dynamic property of AdaBoost.

Although much effort has been spent on understanding how AdaBoost works, questions remain about the success of boosting that are left unanswered [54]. However, on the other hand, the mystery of AdaBoost also implies the room to interpret AdaBoost in a more elegant way and some novel booting-like algorithms could accordingly be derived from the new interpretation.

1.4 Novel totally corrective boosting algorithms and the application

In this thesis, we interpret the success of AdaBoost from the perspective of primal-dual optimization procedure. The novel interpretation also leads to several new boosting algorithms.

Firstly, we provide a insight into the relation between the margin distribution and the generalization capability. In particular, the standard AdaBoost and LPBoost with hard or soft margins are compared in terms of margin distribution and performance. We empirically prove that, for the boosting-like algorithms, the margin distribution plays a more significant role than that of the minimum margin. This observation motivates us to analyze AdaBoost as an optimization procedure over the margin distribution.

Then, from the perspective of convex optimization, we prove that the Lagrange dual problems of AdaBoost, LogitBoost and soft-margin LPBoost with generalized hinge loss are all entropy regularized LPBoost. By employing a general column-generation method, we propose a novel boosting-like algorithm termed AdaBoost-CG. The proffered algorithm minimizes the dual problem of AdaBoost in a column-generation fashion and achieves better performance than its prototype. Furthermore, with some minor assumptions, we derive another new loss function of boosting algorithm that is explicitly related to the margins' distribution. By directly optimizing the distribution, the Margin Distribution Boosting (MDBoost) outperforms other boosting algorithms on most datasets.

The success of the proposed algorithms inspires us to extend the primal–dual conception and the column-generation algorithm to more general cases. A universal framework is designed for minimizing an arbitrary empirical risk function with different regularizations. Under the universal boosting framework, we also develop a faster totally

corrective boosting algorithm by solving the primal problem at each iteration. Various loss functions and regularizations are empirically compared under the framework.

Finally, we apply the totally corrective boosting algorithm to a long-standing problem in computer vision—the face recognition problem. In the past decade, much attention has been paid to an emerging algorithm family, linear representation based face recognition. Lots of important face recognition approaches, such as Nearest Feature Line (NFL) [44], Nearest Feature Subspace (NFS) [13], Sparse Representation Classification (SRC) [98] and Linear Regression Classification (LRC) [57] could be viewed as members of this family, with different prior assumptions. In this thesis, the LRC algorithm is boosted for higher accuracy and robustness. Both the multiple-class boosting algorithm and its totally corrective variation are employed and a remarkable performance improvement is observed. In particular, the totally corrective version illustrate better performance because of its global optimum and higher efficiency thanks to the fewer weak classifiers selected.

We make five major contributions in this thesis, they are:

1. In Chapter 2, we empirically prove that the margin distribution is significant to the explanation of AdaBoost's success. In contrast, too much effort to maximize the minimum margin usually leads to the decline in performance.

2. In Chapter 3, we derive the Lagrangian duals of boosting algorithms and show that most of them are entropy maximization problems. Furthermore, based on the duals we derive, we design column-generation based optimization techniques for boosting learning. We show that the new algorithm, which is termed AdaBoost-CG, has almost identical results to those of standard iterative additive boosting algorithms but with much faster convergence rates. Therefore fewer weak classifiers are needed to build the ensemble.

3. In Chapter 4, we propose another new totally corrective boosting algorithm, MDBoost, that optimizes the margin distribution directly. The optimization procedure of MDBoost is based on the idea of column-generation, which has been widely used in large-scale linear programming. We empirically demonstrate that MDBoost outperforms AdaBoost on most UCI data sets used in our experiments. The success of MDBoost verifies the conjecture in [65].

4. In Chapter 5, we propose a very general framework that can accommodate *arbitrary* convex regularization terms other than ℓ_1 norm. By explicitly deriving the Lagrange dual formulations, we demonstrate that totally corrective boosting

based on column-generation can be designed to facilitate boosting. In particular, we focus on analyzing the ℓ_1, ℓ_2, and ℓ_∞ norm regularization. Besides, we observe that the totally corrective boosting's primal problems are much simpler than the counterpart dual problems. So it is much faster to solve the primal problem, at each iteration of a column-generation based boosting algorithm. In the proposed AnyBoost$_{\text{TC}}$, generally we do not require sophisticated convex solvers and only gradient descent methods like L-BFGS-B [104] are needed. Previous totally corrective boosting algorithms [16, 80, 97] all solve the dual problems using convex optimization solvers.

5. In Chapter 6, we successfully boost the LRC algorithm for face recognition. To our best knowledge, this is the first time that the instance-based algorithm have been boosted to solve the computer vision tasks. In addition, the totally corrective boosting algorithm is also applied to face recognition, achieving even better performance.

Chapter 2

Boosting the minimum margin: LPBoost *vs*. AdaBoost

2.1 Introduction

AdaBoost can be viewed as a gradient descent procedure that minimizes the exponential classification error function. Meanwhile, [28] has proved that the sample margin distribution as well as the minimum margin are reduced aggressively. However, it is not theoretically *perfect* from the viewpoint of optimization since the training is an iterative gradient-descent procedure. Some alternative optimization strategies have been proposed. The linear programming (LP) based boosting, or LPBoost, is one of them. Grove and Schuurmans [35] formulated ensemble learning as an LP to improve the performance of AdaBoost. This approach being theoretically elegant, the authors have observed poorer classification accuracy on test data than AdaBoost. Later, inspired by Grove and Schuurmans' LP formulation and the success of soft-margin support vector machines (SVMs), Demiriz *et al.* [16] introduced soft-margin LP boosting. The column-generation technique is used to solve the LP with a possibly infinite number of weak classifiers. Different from the AdaBoost, the linear coefficients of LPBoost are totally corrective, which in theory results in faster convergence.

Intuitively, LPBoost should have better generalization capability (smaller test error) than AdaBoost, according to the margin theory [74]. LPBoost optimizes the minimum margin directly, while it is not yet clear how the training procedure of AdaBoost maximizes the margin. To date, however, there is no empirical comparison or theoretical explanation of LPBoost against AdaBoost.

In this chapter, we attempt to answer this question. Our results seem *counter-intuitive* at first glance: generally AdaBoost has better generation capability than LP-

Boost. We also give an explanation showing that this observation is in fact consistent with the observation of [65]: maximizing the minimum margin may not be an optimal choice. We also show the connection between LPBoost's cost function and AdaBoost's cost function.

To our knowledge, this is the first work that presents a systematic comparison between LPBoost and AdaBoost. Our analysis also opens the possibility of finding better ways to improve AdaBoost or even of inventing new boosting algorithms.

The closest related work to ours might be [65]. In [65] the authors compared the performance of AdaBoost and Arc-Gv [8]. By comparing the minimum margins and test error of AdaBoost and Arc-Gv, it is empirically shown that larger minimum margins do not necessarily generalize better. Our work confirms this conjecture. In the machine learning community, most work on boosting has been focused on giving new interpretations [29, 53] and exploring new applications [91, 45].

The rest of the chapter is organized as follows. In Section 2.2, we briefly review the concepts of AdaBoost and LPBoost. We then test the two algorithms on various datasets in Section 2.3. After experiments, in Section 2.4, we explain the results by considering the margin distributions. Finally, concluding remarks are given.

2.2 Boosting Algorithms

In this section, we briefly review some boosting-like algorithms, namely AdaBoost, LP-AdaBoost and LPBoost.

2.2.1 AdaBoost

AdaBoost is the first practical and efficient algorithm for ensemble learning [74]. The training procedure of AdaBoost is a greedy algorithm, which constructs an additive combination of weak classifiers such that the exponential loss is minimized:

$$L(y, f(\boldsymbol{x})) = \exp\big(-yF(\boldsymbol{x})\big). \tag{2.1}$$

Here $\boldsymbol{x}$ is the labeled training sample and $y \in \{+1, -1\}$ is its label; $F(\boldsymbol{x})$ is the final decision function which outputs the predicted class label. AdaBoost combines iteratively a number of weak classifiers to form a strong classifier. A weak classifier is defined as a classifier with greater than average accuracy on the training set. The final strong classifier $F(\cdot)$ can be defined as

$$F(\boldsymbol{x}) = \operatorname{sgn}\big(\sum\nolimits_{j=1}^{N} w_j h_j(\boldsymbol{x})\big), \tag{2.2}$$

where w_j is a weight coefficient; $h_j(\cdot)$ is a weak learner with output $\{+1, -1\}$ and N is the number of weak classifiers. At each iteration, AdaBoost selects a new hypothesis $h(\cdot)$ that best classifies training samples with minimal classification error. Each training sample receives a weight that determines its probability of being selected for a training set. If a training sample is correctly classified, then its probability of being used again in a subsequent component classifier is reduced. Conversely, if the pattern is misclassified, then its probability of being used again is increased. In this way, the algorithm focuses more on the misclassified samples after each round of boosting. The AdaBoost algorithm is summarized as follows.

- Given a training set $\{\boldsymbol{x}_1, y_1\}, \ldots, \{\boldsymbol{x}_M, y_M\}$; and initialize $D_1^i = \frac{1}{M}$, (i indexes training samples).
- For $j = 1, \cdots, N$:
  1. Train weak classifier $h_j(\cdot)$ using distribution D_i.
  2. Calculate $w_j = 0.5 \log\left(\frac{1-\epsilon}{\epsilon}\right)$
  3. Update $D_{j+1}^i \propto D_j^i \exp\big(-w_j y_i h_j(\boldsymbol{x}_i)\big)$ and normalize D_{j+1} to be a probability distribution.
- Output the strong classifier as in (2.2).

Schapire *et al.* [74] proved that the larger margins on the training set translate into a superior upper bound on the generalization error. Specifically, using the margins, a bound is shown that does not dependent on the number of boosting rounds, *i.e.*, the generalization error is at most

$$\Pr(\text{margin}(\boldsymbol{x}, y) \leq \Theta) + \tilde{O}\left(\sqrt{\frac{d}{M\Theta^2}}\right) \tag{2.3}$$

for any Θ. Here $\Pr(\cdot)$ denotes the empirical probability on the training samples, M is the size of training samples, and d is the VC-dimension of the space of all possible weak classifiers. The margin of an example is defined as

$$\text{margin}(\boldsymbol{x}, y) = \frac{y(\sum_j w_j h_j(\boldsymbol{x}))}{\sum_j w_j}. \tag{2.4}$$

This result is important because it not only indicates the AdaBoost is somewhat "immune" to overfitting, but also implies that the margin is an important factor affecting the algorithm's classification performance.

2.2.2 LP-AdaBoost

The first contribution on the application of LP to boosting was made by Grove and Schuurmans [35], who derived an LP optimization problem based on maximizing the minimum margin of the combined classifier generated by AdaBoost. Incremental work [62] borrowed the idea from soft-margin SVMs to form a soft-margin LP-AdaBoost. The goal of the LP-AdaBoost is to find the optimal linear weight coefficients that maximize the minimum margin among all the training samples. Mathematically, LP-AdaBoost solves the following LP:

$$\begin{aligned} \max_{\boldsymbol{w},\boldsymbol{\xi},\rho} \quad & \rho - D\sum_{i=1}^{M}\xi_i \\ \text{subject to} \quad & y_i \boldsymbol{h}_i \boldsymbol{w} + \xi_i \geq \rho, \\ & \xi_i \geq 0, i = 1, \cdots, M, \\ & \boldsymbol{w} \geq 0, \mathbf{1}^\top \boldsymbol{w} = 1. \end{aligned} \tag{2.5}$$

Here the variable of interest to be optimized is the minimum margin ρ. This LP formulation is similar to the ℓ_1-norm SVM. $\boldsymbol{w} = [w_1, \cdots, w_N]$ is a column vector. ξ_i is the slack variable for each training example. $\mathbf{1}$ is a column vector, with all entries being 1. Each entry of the vector $\boldsymbol{h}_i = [h_1(\boldsymbol{x}_i), \cdots, h_N(\boldsymbol{x}_i)]$ is the weak classifier $h_j(\cdot)$ response on training example $\boldsymbol{x}_i$. In LP-AdaBoost, the weak classifier set is obtained by a pre-run of AdaBoost. Clearly the trade-off parameter D balances the margin and training error. When $D = 0$, Equation (2.5) is the hard-margin LP-AdaBoost of [35]. In experiments, D is selected by sub-sampling validation.

2.2.3 Column Generation and LPBoost

In many circumstances the cardinality of the F_i, *i.e.* the size of the set of weak classifiers, is too large to cope with by using standard LP solvers. Ideally the set of weak classifiers should not be confined to those generated by AdaBoost. If all the possible weak classifiers are considered, usually one has an infinitely number of weak classifiers. Demiriz *et al.* [16] solved this problem by using the Column-Generation (CG) technique [49, 16]. They termed their new column-generation based boosting LPBoost. The essential idea of LPBoost is identical with LP-AdaBoost while their optimization strategies are different.

Column-generation is a state-of-the-art method for optimally solving difficult large-scale optimization problems. It is a method that avoids considering all variables of a problem explicitly. If an LP has great many variables (columns) but much fewer con-

straints, the column-generation idea can be very beneficial. The crucial insight behind CG is that the number of non-zero variables of the optimal solution is equal to the number of linear constraints, hence although the number of possible variables may be large, we need only a small subset of these in the optimal solution. It works by considering only a small subset of the entire variable set. Once it is solved, we ask the question: "Are there any other variables that can be included to improve the solution?" So we must be able to solve the subproblem: given a set of dual values, one either identifies a variable that has a favorable reduced cost, or indicates that such a variable does not exist. In essence, CG finds the variables with negative reduced costs without explicitly enumerating all variables.

In practice, CG mainly works on the Lagrange dual problem [4]. In the primal space, the CG method solves the problem on a subset of variables, which corresponds to a subset of constraints in the dual. When a column is not included in the primal, the corresponding constraint does not appear in the dual. That is to say, a relaxed version of the dual problem is solved. If a constraint absent from the dual problem is violated by the solution to the restricted problem, this constraint needs to be included in the dual problem to further restrict its feasible region. In other word, instead of adding the "columns" in the primal problem, we augment the constrains one by one to the dual problem.

In the case of LPBoost, Demiriz *et al*. had derived the dual formulation of (2.5) as:

$$\begin{aligned} &\min_{r,\boldsymbol{u}} \ r \\ &\text{s.t. } \textstyle\sum_{i=1}^{M} u_i y_i h_j(\boldsymbol{x}_i) \leq r \ (\forall j = 1, \cdots, N), \\ &D\mathbf{1} \succcurlyeq \boldsymbol{u} \succcurlyeq \mathbf{0}, \mathbf{1}^\top \boldsymbol{u} = 1. \end{aligned} \tag{2.6}$$

To apply the idea of column-generation, one needs to put the constraints successively to the dual problem. Therefore at step t, the associated "sub-problems", *i.e.* the dual problem of the restricted master problem [49], writes

$$\begin{aligned} &\min_{r,\boldsymbol{u}} \ r \\ &\text{s.t. } \textstyle\sum_{i=1}^{M} u_i y_i \hat{h}_j(\boldsymbol{x}_i) \leq r \ (\forall j = 1, \cdots, t), \\ &D\mathbf{1} \succcurlyeq \boldsymbol{u} \succcurlyeq \mathbf{0}, \mathbf{1}^\top \boldsymbol{u} = 1, \end{aligned} \tag{2.7}$$

where $\hat{h}_j(\boldsymbol{x}_i)$ indicates the output of the jth selected weak classifier, w.r.t. the ith training sample. This process stops when no such violated constraint could be found, *i.e.*, the optimality is achieved. Algorithm 1 illustrates the optimization process.

2.2.4 Connection between AdaBoost and LPBoost

As mentioned above, LPBoost is mainly inspired by SVMs, Particularly, ℓ_1-norm SVMs. Is there any connection between LPBoost and AdaBoost? We show that LP-Boost is actually minimizing a bound of AdaBoost's cost function. We know that AdaBoost sequentially solves

$$\min \sum_{i=1}^{M} \exp\left(-y_i F(\boldsymbol{x}_i)\right) \tag{2.8}$$

i.e.,

$$\min \sum_{i=1}^{M} \exp\left(-\text{margin}(\boldsymbol{x}_i, y_i)\right). \tag{2.9}$$

The equivalence holds when $\sum_j w_j = 1$. This problem is the same as

$$\min \log\left(\sum_{i=1}^{M} \exp\left(-\text{margin}(\boldsymbol{x}_i, y_i)\right)\right), \tag{2.10}$$

because $\log(\cdot)$ is strictly monotonically increasing. Therefore we can see that AdaBoost minimizes a log-sum-exp [4] cost function. It is well know that log-sum-exp is a soft differentiable approximation of the $\min(\cdot)$ (and $\max(\cdot)$) function. The following inequality holds:

$$\max_i\{\gamma_i\} \leq \log\left(\sum_{i}^{M} \exp(\gamma_i)\right) \leq \max_i\{\gamma_i\} + \log(M), \tag{2.11}$$

for all γ_i. The second inequality is tight when all components of $\boldsymbol{\gamma}$ are equal. Applying this inequality to (2.10), if we want to minimize the upper bound of the AdaBoost's cost function in (2.10) we have

$$\begin{aligned} & \min \ \max_i\{-\text{margin}(\boldsymbol{x}_i, y_i)\} + \log(M) \\ \rightarrow \quad & \max \ \min_i\{\text{margin}(\boldsymbol{x}_i, y_i)\}. \end{aligned}$$

This is exactly what the hard-margin LP-AdaBoost minimizes.

We have shown that LPBoost actually minimizes the upper bound of AdaBoost's cost function. Considering that the cost function of AdaBoost is already convex, it may not be necessary to minimize an approximation. This may partially explain why LPBoost performs worse than AdaBoost, as shown in our experiments.

2.3 Experiments

2.3.1 Experimental setting

In this section, we perform some experiments to compare the performance between the AdaBoost and the LPBoost. In the experiments, similar to the tests in [16], the LPBoost optimizes the soft-margin cost function, over all the weak hypotheses candidates. Here, we use the standard LP algorithm, rather than the CG version, for the following three reasons. Firstly, according to [65], the complexity of the weak classifier also affects the ensemble's generalization capability. We then employ the decision stump classifier, whose complexity is constant, as the weak learner. Thus in this scenario, the number of the weak classifier candidates is finite. If the dimensionality and the number of training samples are not extremely large, standard LP solvers can solve it without difficulties. Secondly, we have compared the CG based LPBoost and the direct LPBoost, the final generated classifiers are very similar. Finally, directly solving the LP optimization guarantees the global optimum.

As to the experimental data, we choose eight datasets from the UCI machine learning repository. Some of them are also used in [65]. See Table 2.1 for details. Data used for training, validation and test are randomly drawn from the original set. The parameter D in the LPBoost is chosen by using a 10-time repeated sub-sampling validation.

	australian	breast-cancer	fourclass	german	ion	liver	splice	ocr28
number of training data	345	350	431	146	176	173	376	359
number of validation data	173	175	216	73	88	86	250	318
number of test data	172	174	215	72	87	86	250	318

Table 2.1: A description of the UCI datasets used in our experiments.

Both AdaBoost and LPBoost are performed five times on each dataset and we report the average error rates. The experiments are run on a PC with 4G RAM and Intel 2.4G Quad-Core CPU. The software circumstance is Matlab 7.1. We employ the GLPK (GNU Linear Programming Kit) [50] as the LP solver.

2.3.2 Experimental results

We report the experimental results in this section. As is shown in Table 2.2, the competition between AdaBoost and LPBoost seems to end in a draw. Overall, AdaBoost is slightly better. AdaBoost beats LPBoost for the 5 data sets tested in [65]. But LPBoost performs slightly better for the other 3 cases. Thus far, clearly, in terms of the classification performance, LPBoost is not better than AdaBoost.

	australian	breast-cancer	fourclass	german	ion	liver	splice	ocr28
test error (Ada)	0.1651	**0.0472**	0.0808	0.3142	**0.1319**	**0.2934**	**0.1761**	**0.0497**
test error (LP)	**0.1443**	0.0671	**0.0689**	**0.3036**	0.1470	0.2955	0.2123	0.0528
minimum margin (Ada)	0	0.0019	−0.0175	0.0020	0.0043	0.0011	0.0011	0.0365
minimum margin (LP)	−0.2409	0.0105	−0.0985	−0.3194	0.1308	−0.1069	0.0705	0.2724
average margin (Ada)	0.0868	0.2900	0.0760	0.1156	0.2626	0.0761	0.1585	0.5069
average margin (LP)	0.1819	0.2582	0.0792	0.3079	0.1637	0.1088	0.1027	0.3554
D for LP	0.0080	0.1002	0.1260	0.0046	0.1200	0.0242	0.0920	0.2000

Table 2.2: Test results for AdaBoost and soft-magin LPBoost. The last row lists the value of trade-off parameter D for LPBoost. Ada means AdaBoost and LP refers to soft-margin LPBoost.

However, the question is whether LPBoost really benefits from the larger minimum margin? We notice that in the cases AdaBoost fails, its minimum margins are still larger than LPBoost. This observation inspires us to pay a special attention to the soft-margin strategy in LPBoost.

As a result, we conduct the LPBoost again, but with hard margins instead. The results are illustrated in Table 2.3.

	australian	breast-cancer	fourclass	german	ion	liver	splice	ocr28
test error (Ada)	**0.1616**	**0.0471**	**0.1042**	**0.3000**	**0.1563**	**0.2767**	**0.1757**	**0.0497**
test error (LP)	0.2105	0.0609	0.2735	0.3222	0.1586	0.3256	0.2160	0.0541
minimum margin (Ada)	0	0.0015	−0.0234	0.0013	0.0129	0	0.0011	0.0365
minimum margin (LP)	0.0251	0.0577	0.0085	0.0859	0.1406	0.0263	0.0705	0.2742
average margin (Ada)	0.0891	0.2702	0.0771	0.1071	0.3050	0.0705	0.1572	0.5069
average margin (LP)	0.0518	0.2014	0.0469	0.0754	0.1924	0.0485	0.0987	0.3605

Table 2.3: Test results for AdaBoost and hard-magin LPBoost. Note that here LP is hard-margin LPBoost.

Now AdaBoost outperforms LPBoost in all the cases, even though the minimum margins for the LPBoost are consistently larger than for AdaBoost. Therefore, we can draw at least two empirical conclusions. Firstly, the minimum margin is not the key fact affecting the performance in an boosting algorithm. Under some conditions, it does not directly decrease the generalization error. Secondly, it is the soft margin that improves the performance of LPBoost. Without soft margins, the LPBoost is inferior to AdaBoost. In addition, this improvement is based on the price of heavier computation of validation.

We have also reported the average margins here. It seems that the average margin makes more sense for its correlation with the final generalization capability. We will give an explanation for this phenomenon in the next section.

2.4 Discussion

The connection between AdaBoost and LPBoost is derived from the zero–sum game [26, 25], which is used to understand the AdaBoost in a new way*. Since the problem of solving a zero–sum game is well known to be solvable use LP, with the novel understanding of AdaBoost, many attempts like [35] and [16] try to using LP to improve or even replace AdaBoost. They hope the globally optimal solution would lead to better performance. However, no statistically better results have been observed.

Our second experiment (hard-margin LPBoost) proves that the larger minimum margin does not necessarily indicate a better performance. Why does the performance sometimes deteriorate with our attempts to optimize the minimum margin globally? Let us recall the loss functions of AdaBoost and LPBoost. As pointed out in Section 2.2.4, AdaBoost is a procedure for finding a linear combination of base classifiers that attempts to minimize the log-sum-exp cost function. The cost function of AdaBoost is a non-linear convex function. LPBoost actually optimizes the upper bound of AdaBoost's cost function. This replacement of the loss function leads to the performance difference.

LPBoost, especially the hard-margin LPBoost, aims to maximize the minimum margin among all the training samples. This is also proved empirically in the last section. However, AdaBoost, which aims to minimize an sum of the samples' exponential (minus) margins, pays more attention to the margin distribution rather than to the minimum margin. We rerun the experiment again and this time we record the margins for all training data both in AdaBoost and LPBoost.

Figures 2.1 show the cumulative margins of AdaBoost and LPBoost (soft margin) for the 4 cases that AdaBoost wins. We can see that the margin distributions of AdaBoost are all better than LPBoost's distributions, although in terms of the minimum margin, LPBoost is larger than AdaBoost in all 4 cases. This shows that the minimum margin is not as important as we expected. What matters is the entire margin distribution.

During the test, we also compare the margins obtained by the three methods (AdaBoost, hard margin LPBoost and soft-margin LPBoost) for each example and compute the percentage that LPBoost's margin is larger than AdaBoost's margin among all the examples. Let us write this percentage as P_{margin}. The larger P_{margin} indicates a better margin distribution over AdaBoost's margin distributions and generally im-

*In the game-theoretic setting, the AdaBoost algorithm and the weak learner play as two competitors. One tries to minimize the expected loss while the other one tries to maximize it. Please refer to [26, 25] for details.

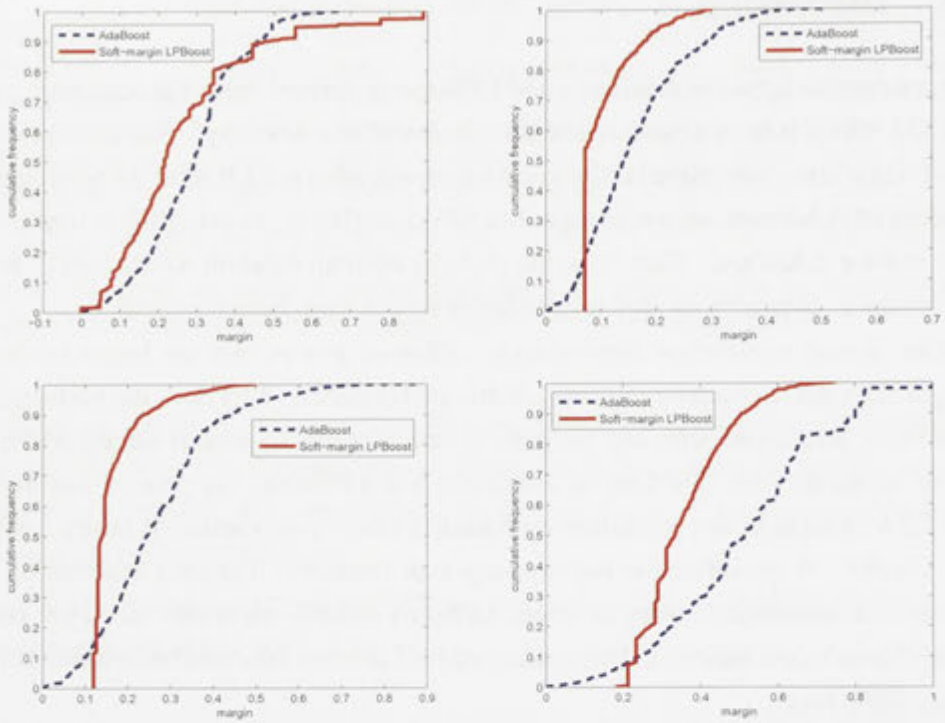


Figure 2.1: The margin distributions of AdaBoost and soft-margin LPBoost on the datasets "breast cancer","splice","ion" and "ocr28". The margin distributions of AdaBoost are all "better" than LPBoost's. Although in terms of the minimum margin, LPBoost is larger than AdaBoost in all the cases.

plies the better performance, as is shown in Table 2.4. From Table 2.4, we can see that for the 3 cases that LPBoost performs better than AdaBoost, their P_{margin} value is larger than half. That means, over half of the margins are larger than AdaBoost's corresponding margins: the overall distribution is better.

	australian	breast-cancer	fourclass	german	ion	liver	splice	ocr28
P_{margin} soft-margin LP	**85.86%**	25.66%	**52.25%**	**61.10%**	17.73%	59.31%	19.68%	17.38%
performace compared to AdaBoost	better	worse	better	better	worse	worse	worse	worse
P_{margin} hard-margin	11.42%	9.49%	12.25%	25.34%	17.16%	19.88%	15.25%	17.86%
performace compared to AdaBoost	worse	worse	worse	worse	worse	worse	worse	worse

Table 2.4: The P_{margin} (percentage of samples with larger margins in LPBoost than those in AdaBoost) and the comparison outcomes (with respect to test error) between LPBoost and AdaBoost.

We show one more example. Figure 2.2 illustrates the cumulative distribution of the margins of AdaBoost, hard-margin LPBoost and soft-margin LPBoost on the dataset *liver disorder*. The soft-margin LPBoost shows the best margin distribution as well as the best performance. The soft-margin strategy, including the validation proce-

dure, ameliorates the margin distribution of LPBoost and helps it win the competition.

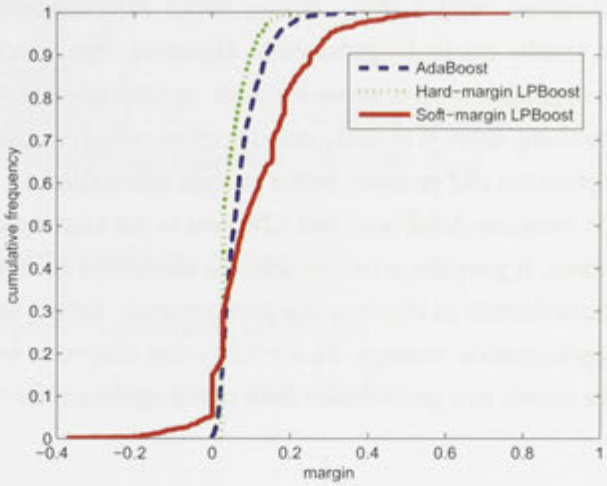


Figure 2.2: The three margin distributions (AdaBoost, hard-margin LPBoost and soft-margin LPBoost) on dataset "liver disorder". In this case, soft-margin LPBoost has the best margin distribution and hence best classification performance.

Here, we summarize the discussion in this section. The margin distribution plays a more important role than the minimum margin in terms of the classification accuracy for boosting algorithms. Using LP to approximate the AdaBoost's loss function will make the new algorithm (LPBoost) neglect to optimize the margin distribution and pay too much attention to the minimum margin, which almost always deteriorates the performance. The soft-margin strategy helps LPBoost to some extend because it can "shift" the attention from the minimum margin to the training error and may lead to a better margin distribution than hard-margin LPBoost has. But still it is hard to beat AdaBoost.

2.5 Conclusion

The margin distribution is significant to the explanation of AdaBoost's success. Many researchers have noticed this phenomenon. AdaBoost's loss function ensures that the margin distribution will be better and better iteratively, while LPBoost scarifies most instances' margins to increase the minimum one. The gradient-based feature is a drawback of AdaBoost. The LPBoost amends this drawback but at the same time, it brings a new and even bigger drawback. Our analysis shows that using LPBoost to approximate AdaBoost is improper and inefficient.

Many researchers are still focusing on ameliorating AdaBoost as well as LPBoost, but most of them consider the AdaBoost as a simple max–min game or pay too much attention to the minimum margin [64]. If they move their interest into the margin distribution, better results might be produced. However, the direction of replacing the gradient-based approach with a more efficient optimization technique is correct undoubtedly. A promising topic is to find a cost function, which would better be convex (and thus easy to optimize) and produce better margin distributions.

The comparison between AdaBoost and LPBoost is the starting point of our work on boosting algorithms. It provides a insight into the success of boosting algorithm and implies two important factors to improve the performance: better margin distribution and more elegant optimization strategy. In the following chapters, we will extend this idea and achieve the above two goals under the convex optimization framework.

Algorithm 1 LPBoost

Input: Training set $(\boldsymbol{x}_i, y_i), i = 1 \cdots M$; termination threshold $\varepsilon > 0$; regularization parameter D; (optional) maximum iteration $N_{\max}$.

Initialization:

1. $N = 0$ (no weak classifiers selected);
2. $\boldsymbol{w} = 0$ (all primal coefficients are zeros);
3. $u_i = \frac{1}{M}, i = 1 \cdots M$ (uniform dual weights).

while true **do**

1. Find a new base $h'(\cdot)$ which has the lowest training error w.r.t. the current data distribution $\boldsymbol{u} = [u_1, u_2, \cdots, u_M]$.
2. Check for optimal solution:
   if $\sum_{i=1}^{M} u_i y_i h'(\boldsymbol{x}_i) < r + \varepsilon$, **then** break (problem solved);
3. Add $h'(\cdot)$ to the restricted master problem, which corresponds to a new constraint in the dual;
4. Solve the dual problem (2.7) to obtain updated r and u_i $(i = 1, \cdots, M)$;
5. $N = N + 1$ (weak classifier count);
6. (optional) **if** $N \geq N_{\max}$, **then** break (maximum iteration reached).

end

Output:

1. Calculate the primal variable $\boldsymbol{w}$ from the optimality conditions and the last solved dual problem;
2. The learned classifier $F(\boldsymbol{x}) = \sum_{j=1}^{N} w_j h_j(\boldsymbol{x})$.

Chapter 3

On the Dual Formulation of Boosting Algorithms

3.1 Introduction

As introduced in Chapter 2, LPBoost was invented by [35, 16] with the intuition of maximizing the minimum margin of all training examples. The final optimization problem can be formulated as a linear program (LP). We also observed that the hard-margin LPBoost does not perform well in most cases although it usually produces larger minimum margins. In other words, a higher minimum margin would not necessarily imply a lower test error. Breiman [8] also noticed the same phenomenon: his Arc-Gv algorithm has a minimum margin that provably converges to the optimal, but Arc-Gv is inferior in terms of generalization capability. Experiments on LPBoost and Arc-Gv have put the margin theory into serious doubt. Until recently, Reyzin and Schapire [65] re-ran Breiman's experiments by controlling weak classifiers' complexity. They found that the minimum margin is indeed larger for Arc-Gv, but the overall margin distribution is typically better for AdaBoost. The conclusion is that the minimum margin is important, but not always at the expense of other factors. They also conjectured that maximizing the average margin, instead of the minimum margin, may result in better boosting algorithms.

As the soft-margin SVM usually has a better classification accuracy than the hard-margin SVM, the soft-margin LPBoost also performs better by relaxing the constraints that all training examples must be correctly classified. Cross-validation is required to determine an optimal value for the soft-margin trade-off parameter. The equivalence between SVMs and boosting-like algorithms is shown in [61]. Comprehensive overviews on boosting are given by [55] and [75].

We show in this chapter that the Lagrange duals of AdaBoost, LogitBoost and LPBoost with generalized hinge loss are all entropy maximization problems. Previous work like [14, 41, 42] noticed the connection between boosting techniques and entropy maximization based on Bregman distances. They did not show that the duals of boosting algorithms are actually entropy regularized LPBoost as we show in (3.11), (3.29) and (3.32). In addition, no column-generation based optimization algorithms for AdaBoost or LogitBoost were derived in the literature. By knowing this duality equivalence, we derive a general column-generation based optimization framework that can be used to optimize arbitrary convex loss functions. In other words, we can easily design totally corrective AdaBoost, LogitBoost and boosting with generalized hinge loss, *etc*.

Our major contributions are the following:

1. We derive the Lagrangian duals of boosting algorithms and show that most of them are entropy maximization problems.

2. The authors of [65] conjectured that "it may be fruitful to consider boosting algorithms that greedily maximize the average or median margin rather than the minimum one". We theoretically prove that, actually, AdaBoost approximately maximizes the average margin, instead of the minimum margin. This is an important result, in the sense that it provides an alternative theoretical explanation that is both consistent with the margins theory and that agrees with the empirical observations made by [65].

3. We propose AdaBoost-QP that directly optimizes the asymptotic cost function of AdaBoost. The experiments confirm our theoretical analysis.

4. Furthermore, based on the duals we derive, we design column-generation based optimization techniques for boosting learning. We show that the new algorithms have almost identical results to those of standard stage-wise additive boosting algorithms but with much faster convergence rates. Therefore fewer weak classifiers are needed to build the ensemble.

The chapter is organized as follows. Section 3.2 briefly reviews several boosting algorithms for self-completeness. Their corresponding duals are derived in Section 3.3. Our main results are also presented in Section 3.3. In Section 3.4, we then present numerical experiments to illustrate various aspects of our new algorithms obtained in Section 3.3. We conclude the chapter in the last section.

3.2 The Loss Functions of Boosting Algorithms

We first review some basic ideas, and the corresponding optimization problems of AdaBoost, LPBoost and LogitBoost, which are of interest in this present work.

Let $\mathcal{H}$ be a class of base classifier $\mathcal{H} = \{h_j(\cdot) : \mathcal{X} \to \mathbb{R}, j = 1 \cdots N\}$. A boosting algorithm seeks for a convex linear combination

$$F(\boldsymbol{w}) = \sum_{j=1}^{N} w_j h_j(\boldsymbol{x}), \tag{3.1}$$

where $\boldsymbol{w}$ is the weak classifier weights to be optimized. AdaBoost calls an oracle that selects a weak classifier $h_j(\cdot)$ at each iteration j and then calculates the weight w_j associated with $h_j(\cdot)$. It is shown in [29, 52] that AdaBoost (and many others such as LogitBoost) performs coordinate gradient descent in function space, at each iteration choosing a weak classifier to include in the combination, such that the cost function is maximally reduced. It is well known that coordinate descent has a slow convergence in many cases. From an optimization point of view, there is no particular reason to keep the weights $w_1, \cdots, w_{j-1}$ fixed at iteration j. Here we focus on the underlying mathematical programs that boosting algorithms minimize.

AdaBoost has been proved to minimize the exponential loss function [14]:

$$\min_{\boldsymbol{w}} \sum_{i=1}^{M} \exp(-y_i F(\boldsymbol{x}_i)), \text{ s.t. } \boldsymbol{w} \succcurlyeq 0. \tag{3.2}$$

Because the logarithmic function $\log(\cdot)$ is a strictly monotonically increasing function, AdaBoost equivalently solves

$$\min_{\boldsymbol{w}} \log\left(\sum_{i=1}^{M} \exp(-y_i F(\boldsymbol{x}_i))\right), \text{ s.t. } \boldsymbol{w} \succcurlyeq 0, \mathbf{1}^\top \boldsymbol{w} = \tfrac{1}{\mathrm{T}}. \tag{3.3}$$

Note that in the AdaBoost algorithm, the constraint $\mathbf{1}^\top \boldsymbol{w} = \frac{1}{\mathrm{T}}$ is not explicitly enforced. However, without this regularization constraint, one can always make the cost function approach zero via enlarging the solution $\boldsymbol{w}$ by an arbitrarily large factor. Here what matters is the sign of the classification evaluation function. AdaBoost selects the value of T by selecting how many iterations it runs. We can also use a relaxed version $\mathbf{1}^\top \boldsymbol{w} \leq \frac{1}{\mathrm{T}}$.*

We will show that it is very important to introduce this new cost function. All of our main results on AdaBoost are obtained by analyzing this logarithmic cost function, not AdaBoost's original cost function. Let us define the matrix $H \in \mathbb{Z}^{M \times N}$, which

*The reason why we do not write this constrain as $\mathbf{1}^\top \boldsymbol{w} = \mathrm{T}$ will become clear later.

contains all the possible predictions of the training data using weak classifiers from the pool $\mathcal{H}$. Explicitly $H_{ij} = h_j(\boldsymbol{x}_i)$ is the label ($\{+1, -1\}$) given by weak classifier $h_j(\cdot)$ on the training example $\boldsymbol{x}_i$. We use $H_i = [H_{i1}\ H_{i1} \cdots H_{iN}]$ to denote i-th row of H, which constitutes the output of all the weak classifiers on the training example $\boldsymbol{x}_i$. The cost function of AdaBoost writes:

$$\min_{\boldsymbol{w}} \ \log\left(\sum_{i=1}^{M} \exp(-y_i H_i \boldsymbol{w})\right), \ \text{s.t.} \ \boldsymbol{w} \succcurlyeq \mathbf{0}, \mathbf{1}^\top \boldsymbol{w} = \tfrac{1}{\mathrm{T}}. \tag{3.4}$$

We can also write the above program into

$$\min_{\boldsymbol{w}} \ \log\left(\sum_{i=1}^{M} \exp\left(-\frac{y_i H_i \boldsymbol{w}}{T}\right)\right), \ \text{s.t.} \ \boldsymbol{w} \succcurlyeq \mathbf{0}, \mathbf{1}^\top \boldsymbol{w} = 1, \tag{3.5}$$

which is exactly the same as (3.4). In [71] the smooth margin that is similar but different to the logarithmic cost function, is used to analyze AdaBoost's convergence behavior.[†]

Problem (3.4) (or (3.5)) is a convex problem in $\boldsymbol{w}$. We know that the log-sum-exp function $\mathrm{lse}(\boldsymbol{x}) = \log(\sum_{i=1}^{M} \exp x_i)$ is convex [4]. Composition with an affine mapping preserves convexity. Therefore, the cost function is convex. The constraints are linear and hence also convex. For completeness, we include the description of the standard stage-wise AdaBoost and Arc-Gv in Algorithm 2. The only difference between these two algorithms is the way to calculate w_j (step (2) of Algorithm 2):

$$w_j = \frac{1}{2} \log \frac{1 + r_j}{1 - r_j}, \tag{3.6}$$

where r_j is the edge of the weak classifier $h_j(\cdot)$ defined as $r_j = \sum_{i=1}^{M} u_i y_i h_j(\boldsymbol{x}_i) = \sum_{i=1}^{M} u_i y_i H_{ij}$. Arc-Gv modifies Equ. (3.6) in order to maximize the minimum margin:

$$w_j = \frac{1}{2} \log \frac{1 + r_j}{1 - r_j} - \frac{1}{2} \log \frac{1 + \mathrm{margin}_j}{1 - \mathrm{margin}_j}, \tag{3.7}$$

where margin_j is the minimum margin over all training examples of the combined classifier up to the current round: $\mathrm{margin}_j = \min_i \{y_i \sum_{s=1}^{j-1} w_s h_s(\boldsymbol{x}_i) / \sum_{s=1}^{j-1} w_s\}$, with $\mathrm{margin}_1 = 0$. Arc-Gv clips w_j into $[0, 1]$ by setting $w_j = 1$ if $w_j > 1$ and $w_j = 0$ if $w_j < 0$ [8].

[†]The smooth margin in [71] is defined as

$$\frac{-\log(\sum_{i=1}^{M} \exp(-y_i H_i \boldsymbol{w}))}{\mathbf{1}^\top \boldsymbol{w}}.$$

Algorithm 2 Stage-wise AdaBoost, and Arc-Gv.

Input: Training set $(\boldsymbol{x}_i, y_i), y_i = \{+1, -1\}, i = 1 \cdots M$; maximum iteration $N_{\max}$.

Initialization: $u_i^0 = \frac{1}{M}, \forall i = 1 \cdots M$.

for $j = 1, \cdots, N_{\max}$ **do**

1. Find a new base $h_j(\cdot)$ using the distribution $\boldsymbol{u}^j$;
2. Choose w_j;
3. Update $\boldsymbol{u}$: $u_i^{j+1} \propto u_i^j \exp\left(-y_i w_j h_j(\boldsymbol{x}_i)\right), \forall i$; and normalize $\boldsymbol{u}^{j+1}$.

end

Output: The learned classifier $F(\boldsymbol{x}) = \sum_{j=1}^{N} w_j h_j(\boldsymbol{x})$.

The only difference between LogitBoost and AdaBoost is that LogitBoost adopts a logistic loss rather than an exponential loss. LogitBoost can be formulated as

$$\min_{\boldsymbol{w}} \sum_{i=1}^{M} \log\left(1 + \exp(-y_i H_i \boldsymbol{w})\right), \text{ s.t. } \boldsymbol{w} \succcurlyeq 0, \mathbf{1}^\top \boldsymbol{w} = \tfrac{1}{\mathrm{T}}. \tag{3.8}$$

Similarly this is equivalent to

$$\min_{\boldsymbol{w}} \sum_{i=1}^{M} \log\left(1 + \exp\left(-\frac{y_i H_i \boldsymbol{w}}{T}\right)\right), \text{ s.t. } \boldsymbol{w} \succcurlyeq 0, \mathbf{1}^\top \boldsymbol{w} = 1. \tag{3.9}$$

The logistic loss can be regarded as a smooth hinge loss. It is also convex.

3.3 Lagrange Dual of Boosting Algorithms

We are ready to derive the Lagrange dual of AdaBoost. Our main derivations are based on a form of duality termed the convex conjugate or Fenchel duality.

Definition 3.3.1. *(Convex Conjugate) Let $f : \mathbb{R}^n \to \mathbb{R}$. The function $f^* : \mathbb{R}^n \to \mathbb{R}$, defined as*

$$f^*(\boldsymbol{u}) = \sup_{\boldsymbol{x} \in \operatorname{dom} f} \left(\boldsymbol{u}^\top \boldsymbol{x} - f(\boldsymbol{x})\right), \tag{3.10}$$

is called the convex conjugate (or Fenchel duality) of the function $f(\cdot)$. The domain of the conjugate function consists of $\boldsymbol{u} \in \mathbb{R}^n$ for which the supremum is finite.

$f^*(\cdot)$ is always a convex function because it is the pointwise supremum of a family of affine functions of $\boldsymbol{u}$. This is true even if $f(\cdot)$ is non-convex [4].

Proposition 3.3.1. *(Conjugate of log-sum-exp) The conjugate of the log-sum-exp function is a negative entropy function, restricted to the probability simplex. Formally, for* $\mathrm{lse}(\boldsymbol{x}) = \log(\sum_{i=1}^{M} \exp x_i)$*, its conjugate is:*

$$\mathrm{lse}^*(\boldsymbol{u}) = \begin{cases} \sum_{i=1}^{M} u_i \log u_i, & \text{if } \boldsymbol{u} \succcurlyeq 0 \text{ and } 1^\top \boldsymbol{u} = 1; \\ \infty & \text{otherwise.} \end{cases}$$

We interpret $0 \log 0$ *as* 0*.*

Chapter 3.3 of [4] gives this result.

Theorem 3.1. *The dual of AdaBoost is a Shannon entropy maximization problem, which writes,*

$$\begin{aligned} \max_{r,\boldsymbol{u}} \ & \frac{r}{T} - \textstyle\sum_{i=1}^{M} u_i \log u_i \\ \text{s.t.} \ & \textstyle\sum_{i=1}^{M} u_i y_i H_i \preccurlyeq -r 1^\top, \\ & \boldsymbol{u} \succcurlyeq 0, 1^\top \boldsymbol{u} = 1. \end{aligned} \tag{3.11}$$

Proof: To derive a Lagrange dual of AdaBoost, we first introduce a new variable $\boldsymbol{z} \in \mathbb{R}^M$ such that its i-th entry $z_i = -y_i H_i \boldsymbol{w}$, to obtain the equivalent problem

$$\begin{aligned} \min_{\boldsymbol{w}} \ & \log\left(\textstyle\sum_{i=1}^{M} \exp z_i\right) \\ \text{s.t.} \ & z_i = -y_i H_i \boldsymbol{w} \ (\forall i = 1, \cdots, M), \\ & \boldsymbol{w} \succcurlyeq 0, 1^\top \boldsymbol{w} = \tfrac{1}{\mathrm{T}}. \end{aligned} \tag{3.12}$$

The Lagrangian $L(\cdot)$ associated with the problem (3.4) is

$$\begin{aligned} L(\boldsymbol{w}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{q}, r) = & \log\left(\sum_{i=1}^{M} \exp z_i\right) - \sum_{i=1}^{M} u_i (z_i + y_i H_i \boldsymbol{w}) \\ & - \boldsymbol{q}^\top \boldsymbol{w} - r(1^\top \boldsymbol{w} - \tfrac{1}{\mathrm{T}}), \end{aligned} \tag{3.13}$$

with $\boldsymbol{q} \succcurlyeq 0$. The dual function is

$$\begin{aligned} \inf_{\boldsymbol{z},\boldsymbol{w}} L = & \inf_{\boldsymbol{z},\boldsymbol{w}} \log\left(\sum_{i=1}^{M} \exp z_i\right) - \sum_{i=1}^{M} u_i z_i + \frac{r}{T} \\ & - \overbrace{\left(\sum_{i=1}^{M} u_i y_i H_i + \boldsymbol{q}^\top + r 1^\top\right)}^{\text{must be } \mathbf{0}} \boldsymbol{w} \end{aligned}$$

$$
\begin{aligned}
&= \inf_{\boldsymbol{z}} \log\left(\sum_{i=1}^{M} \exp z_i\right) - \boldsymbol{u}^\top \boldsymbol{z} + \frac{r}{T} \\
&= \overbrace{-\sup_{\boldsymbol{z}} \left[\boldsymbol{u}^\top \boldsymbol{z} - \log\left(\sum_{i=1}^{M} \exp z_i\right)\right]}^{-\text{lse}^*(\boldsymbol{u}) \text{ (see Proposition 3.3.1)}} + \frac{r}{T} \\
&= -\sum_{i=1}^{M} u_i \log u_i + \frac{r}{T}. \qquad (3.14)
\end{aligned}
$$

By collecting all the constraints and eliminating $\boldsymbol{q}$, the dual of Problem (3.4) is (3.11). □

Keeping two variables $\boldsymbol{w}$ and $\boldsymbol{z}$, and introducing new equality constraints $z_i = -y_i H_i \boldsymbol{w}$, $\forall i$, is essential to derive the above simple and elegant Lagrange dual. Simple equivalent reformulations of a problem can lead to very different dual problems. Without introducing new variables and equality constraints, one would not be able to obtain (3.11). Here we have considered the *negative margin* z_i to be the central object of study. In [66], a similar idea has been used to derive different duals of kernel methods, which leads to the so-called *value regularization*. We focus on boosting algorithms instead of kernel methods in this work. Also note that we would have the following dual if we work directly on AdaBoost's cost function in (3.2):

$$
\begin{aligned}
&\max_{r,\boldsymbol{u}} \ \frac{r}{T} - \textstyle\sum_{i=1}^{M} u_i \log u_i + \mathbf{1}^\top \boldsymbol{u} \\
&\text{s.t.} \ \textstyle\sum_{i=1}^{M} u_i y_i H_i \preccurlyeq -r\mathbf{1}^\top, \boldsymbol{u} \succcurlyeq 0. \qquad (3.15)
\end{aligned}
$$

No normalization requirement $\mathbf{1}^\top \boldsymbol{u} = 1$ is imposed. Instead, $\mathbf{1}^\top \boldsymbol{u}$ works as a regularization term. The connection between AdaBoost and LPBoost is not clear with this dual.

Lagrange duality between problems (3.4) and (3.11) assures that weak duality and strong duality hold. Weak duality shows that any feasible solution of (3.11) produces a lower bound of the original problem (3.4). Strong duality tells us that the optimal value of (3.11) is the same as the optimal value of (3.4). The weak duality is guaranteed by the Lagrange duality theory. The strong duality holds since the primal problem (3.4) is a convex problem that satisfies Slater's condition [4].

To show the connection with LPBoost, we equivalently rewrite the above formulation by reversing the sign of r and multiplying the cost function with T, $(T > 0)$:

$$
\begin{aligned}
&\min_{r,\boldsymbol{u}} \ r + T\textstyle\sum_{i=1}^{M} u_i \log u_i \\
&\text{s.t.} \ \textstyle\sum_{i=1}^{M} u_i y_i H_{ij} \leq r \ (\forall j = 1, \cdots, N), \qquad (3.16)
\end{aligned}
$$

$$\boldsymbol{u} \succcurlyeq 0, \mathbf{1}^\top \boldsymbol{u} = 1.$$

Note that the constrain $\boldsymbol{u} \succcurlyeq 0$ is implicitly enforced by the logarithmic function and thus it can be dropped when one solves (3.16). Note, if we have used the inequality regularization constraint $\mathbf{1}^\top \boldsymbol{w} \leq \frac{1}{\Gamma}$, we will have one extra constraint $r \geq 0$ in (3.16).

3.3.1 Connection between AdaBoost and Gibbs free energy

Gibbs free energy is the chemical potential that is minimized when a system reaches equilibrium at constant pressure and temperature.

Let us consider a system that has M states at temperature T. Each state has energy v_i and probability u_i of likelihood of occurring. The Gibbs free energy of this system is related with its average energy and entropy, namely:

$$G(\boldsymbol{v}, \boldsymbol{u}) = \boldsymbol{u}^\top \boldsymbol{v} + T \sum_{i=1}^{M} u_i \log u_i. \tag{3.17}$$

When the system reaches equilibrium, $G(\boldsymbol{v}, \boldsymbol{u})$ is minimized. So we have

$$\min_{\boldsymbol{u}} \; G(\boldsymbol{v}, \boldsymbol{u}), \;\; \text{s.t.} \;\; \boldsymbol{u} \succcurlyeq 0, \mathbf{1}^\top \boldsymbol{u} = 1. \tag{3.18}$$

The constraints ensure that $\boldsymbol{u}$ is a probability distribution.

Now let us define vector $\boldsymbol{v}_j$ with its entries being $v_{ij} = y_i H_{ij}$. v_{ij} is the energy associated with state i for case j. v_{ij} can only take discrete binary values $+1$ or -1. We rewrite our dual optimization problem (3.16) into

$$\begin{aligned} \min_{\boldsymbol{u}} \quad & \overbrace{\max_j \{\boldsymbol{u}^\top \boldsymbol{v}_j\}}^{\text{worst case energy vector } \boldsymbol{v}_j} + T \sum_{i=1}^{M} u_i \log u_i, \\ \text{s.t.} \quad & \boldsymbol{u} \succcurlyeq 0, \mathbf{1}^\top \boldsymbol{u} = 1. \end{aligned} \tag{3.19}$$

This can be interpreted as finding the minimum Gibbs free energy for the *worst* case energy vector.

3.3.2 Connection between AdaBoost and LPBoost: a primal-dual interpretation

First let us recall the basic concepts of LPBoost. The idea of LPBoost is to maximize the minimum margin because it is believed that the minimum margin plays a critically important role in terms of generalization error [77]. The hard-margin LPBoost [35] can be formulated as

$$\max_{\boldsymbol{w}} \; \overbrace{\min_i \{y_i H_i \boldsymbol{w}\}}^{\text{minimum margin}}, \;\; \text{s.t.} \;\; \boldsymbol{w} \succcurlyeq 0, \mathbf{1}^\top \boldsymbol{w} = 1. \tag{3.20}$$

This problem can be solved as an LP. Its dual is also an LP:

$$\begin{aligned} \min_{r,\boldsymbol{u}} \; & r \\ \text{s.t.} \; & \textstyle\sum_{i=1}^{M} u_i y_i H_{ij} \leq r \; (\forall j = 1, \cdots, N), \\ & \boldsymbol{u} \succcurlyeq 0, 1^\top \boldsymbol{u} = 1. \end{aligned} \tag{3.21}$$

Arc-Gv has been shown asymptotically to a solution of the above LPs [8].

The performance deteriorates when no linear combination of weak classifiers can be found that separates the training examples. By introducing slack variables, we get the soft-margin LPBoost algorithm

$$\begin{aligned} \max_{\boldsymbol{w},\varrho,\boldsymbol{\xi}} \; & \varrho - D 1^\top \boldsymbol{\xi} \\ \text{s.t.} \; & y_i H_i \boldsymbol{w} \geq \varrho - \xi_i, (\forall i = 1, \cdots, M), \\ & \boldsymbol{w} \succcurlyeq 0, 1^\top \boldsymbol{w} = 1, \boldsymbol{\xi} \succcurlyeq 0. \end{aligned} \tag{3.22}$$

Here D is a trade-off parameter that controls the balance between training error and margin maximization. The dual of (3.22) is similar to the hard-margin case, except that the dual variable $\boldsymbol{u}$ is capped:

$$\begin{aligned} \min_{r,\boldsymbol{u}} \; & r \\ \text{s.t.} \; & \textstyle\sum_{i=1}^{M} u_i y_i H_{ij} \leq r \; (\forall j = 1, \cdots, N), \\ & D1 \succcurlyeq \boldsymbol{u} \succcurlyeq 0, 1^\top \boldsymbol{u} = 1. \end{aligned} \tag{3.23}$$

Comparing (3.16) with hard-margin LPBoost's dual, it is easy to see that the only difference is the entropy term in the cost function. If we set $T = 0$, (3.16) reduces to the hard-margin LPBoost. In this sense, we can view AdaBoost's dual as entropy regularized hard-margin LPBoost. Since the regularization coefficient T is always positive, the effects of the entropy regularization term is to encourage the distribution $\boldsymbol{u}$ as uniform as possible (the negative entropy $\sum_{i=1}^{M} u_i \log u_i$ is the Kullback-Leibler distance between $\boldsymbol{u}$ and the uniform distribution). This may explain the underlying reason of AdaBoost's success over hard-margin LPBoost: to limit the weight distribution $\boldsymbol{u}$ leads to better generalization performance. But *why and how*? We will discover the mechanism in Section 3.3.3.

When the regularization coefficient, T, is sufficiently large, the entropy term in the cost function dominates. In this case, all discrete probability u_i become almost the same and therefore gather around the center of the simplex $\{\boldsymbol{u} \succcurlyeq 0, 1^\top \boldsymbol{u} = 1\}$. As T decreases, the solution will gradually shift to the boundaries of the simplex to find the

mixture that best approximates the maximum. Therefore, T can also be viewed as a homotopy parameter that bridges a maximum entropy problem with uniform distribution $u_i = 1/M$ $(i = 1, ..., M)$, to a solution of the max-min problem (3.20).

This observation is also consistent with the soft-margin LPBoost. We know that soft-margin LPBoost often outperforms hard-margin LPBoost [35, 16]. In the primal, it is usually explained that the hinge loss of soft-margin is more appropriate for classification. The introduction of slack variables in the primal actually results in box constraints on the weight distribution in the dual. In other words the ℓ_∞ norm of $\boldsymbol{u}$, $\|\boldsymbol{u}\|_\infty$, is capped. This capping mechanism is *harder* than the entropy regularization mechanism of AdaBoost. Nevertheless, both are beneficial on inseparable data. In [78], it is proved that soft-margin LPBoost actually maximizes the average of $1/D$ smallest margins.

Now let us take a look at the cost function of AdaBoost and LPBoost in the primal. The log-sum-exp cost employed by AdaBoost can be viewed as a smooth approximation of the maximum function because of the following inequality:

$$\max_i a_i \leq \log\left(\sum_{i=1}^M \exp a_i\right) \leq \max_i a_i + \log M.$$

Therefore, LPBoost uses a hard maximum (or minimum) function while AdaBoost uses a soft approximation of the maximum (minimum) function. Next, we try to explain why AdaBoost's soft cost function is better than LPBoost's[‡] hard cost function .

3.3.3 AdaBoost controls the margin variance via maximizing the entropy of the weights on the training examples

In AdaBoost training, there are two sets of weights: the weights of the weak classifiers $\boldsymbol{w}$ and the weights on the training examples $\boldsymbol{u}$. In the last section, we assume that to limit $\boldsymbol{u}$ is beneficial for classification performance. By looking at the Karush-Kuhn-Tucker (KKT) conditions of the convex program that we have formulated, we are able to reveal the relationship between the two sets of weights. More precisely, we show how AdaBoost controls the margin variance by optimizing the entropy of weights $\boldsymbol{u}$.

Recall that we have to introduce new equalities $z_i = -y_i H_i \boldsymbol{w}, \forall i$ in order to obtain the dual (3.11) (and (3.16)). Obviously z_i is the negative margin of sample $\boldsymbol{x}_i$. Notice that the Lagrange multiplier $\boldsymbol{u}$ is associated with these equalities. Let $(\boldsymbol{w}^\star, \boldsymbol{z}^\star)$ and $(\boldsymbol{u}^\star, \boldsymbol{q}^\star, r^\star)$ be any primal and dual optimal points with zero duality gap. One of the

[‡]Hereafter, we use LPBoost to denote hard-margin LPBoost unless otherwise specified.

KKT conditions tells us

$$\nabla_{\boldsymbol{z}} L(\boldsymbol{w}^\star, \boldsymbol{z}^\star, \boldsymbol{u}^\star, \boldsymbol{q}^\star, r^\star) = 0. \tag{3.24}$$

The Lagrangian $L(\cdot)$ is defined in (3.13). This equation follows

$$u_i^\star = \frac{\exp z_i^\star}{\sum_{i=1}^M \exp z_i^\star}, \ \forall i = 1, \cdots M. \tag{3.25}$$

Equ. (3.25) guarantees that $\boldsymbol{u}^\star$ is a probability distribution. Note that (3.25) is actually the same as the update rule used in AdaBoost. The optimal value[§] of the Lagrange dual problem (3.11), which we denote $\mathrm{Opt}^\star_{(3.11)}$, equals the optimal value of the original problem (3.4) (and (3.12)) due to the strong duality, hence $\mathrm{Opt}^\star_{(3.4)} = \mathrm{Opt}^\star_{(3.11)}$.

From Equ. (3.25), at optimality we have

$$\begin{aligned} -z_i^\star &= -\log u_i^\star - \log\big(\textstyle\sum_{i=1}^M \exp z_i^\star\big) \\ &= -\log u_i^\star - \mathrm{Opt}^\star_{(3.11)} \\ &= -\log u_i^\star - \mathrm{Opt}^\star_{(3.4)}, \ \forall i = 1, \cdots M. \end{aligned} \tag{3.26}$$

This equation suggests that, after convergence, the margins' values are determined by the weights on the training examples $\boldsymbol{u}^\star$ and the cost function's value. From (3.26), the margin's variance is entirely determined by $\boldsymbol{u}^\star$:

$$\mathrm{var}\{-\boldsymbol{z}^\star\} = \mathrm{var}\{\log \boldsymbol{u}^\star\} + \mathrm{var}\{\mathrm{Opt}^\star_{(3.4)}\} = \mathrm{var}\{\log \boldsymbol{u}^\star\}. \tag{3.27}$$

We now understand the reason why capping $\boldsymbol{u}$, as LPBoost does, or uniforming $\boldsymbol{u}$ as AdaBoost does can decrease the margins' deviation. These two equations reveal the important role that the weight distribution $\boldsymbol{u}$ plays in AdaBoost. All that we knew previously was that the weights on the training examples measure how difficult it is to correctly classify an individual example. In fact, besides that, the weight distribution on the training examples is also a *proxy* for minimizing the margin's distribution divergence. From the viewpoint of optimization, this is an interesting finding. In AdaBoost, the main purpose is to control the divergence of the margin distribution, which may not be easy to optimize *directly* because a margin can take a value out of the range $[0, 1]$, where entropy is not applicable. AdaBoost's cost function allows one to do so *implicitly* in the primal but *explicitly* in the dual. This is a stimulating technique. A future research topic is to apply this idea to other machine learning problems.

The connection between the dual variable $\boldsymbol{u}$ and margins tells us that AdaBoost does not only optimize the minimum margin[¶] but also takes the variance of the margins

[§]Hereafter we use the symbol $\mathrm{Opt}^\star_{(\cdot)}$ to denote the optimal value of Problem $(\cdot)$.

[¶]Or average margin? We will answer this question in the next section.

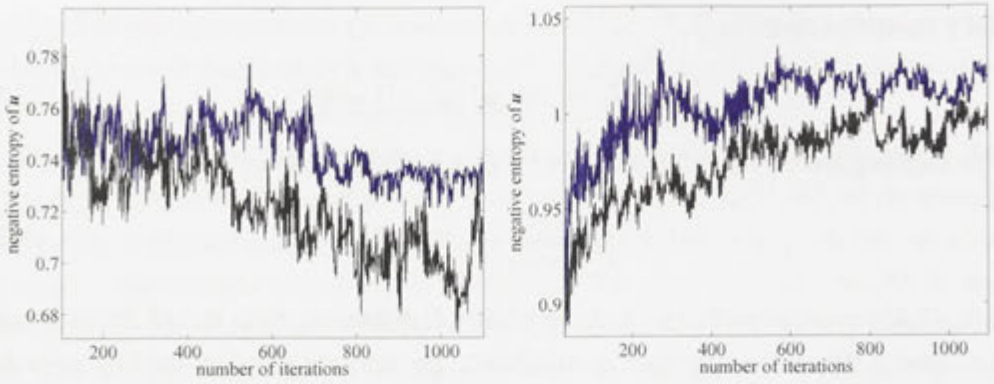


Figure 3.1: Negative entropy of $\boldsymbol{u}$ produced by AdaBoost and Arc-Gv at each iteration on datasets *breast-cancer* and *australian* respectively. The negative entropy produced by AdaBoost (black) is consistently lower than the one by Arc-Gv (blue).

into consideration. In the dual problem (3.16), minimizing the maximum edge on the weak classifiers contributes to maximizing the margin. At the same time, minimizing the negative entropy of weights on training examples contributes to controlling the margin's variance. We make this useful observation by examining the dual problem as well as the KKT optimality conditions. But it remains unclear about the exact statistical measures that AdaBoost optimizes. The next section presents a complete answer to this question through analyzing AdaBoost's primal optimization problem.

We know that Arc-Gv chooses $\boldsymbol{w}$ in a different way from AdaBoost. Therefore Arc-Gv optimizes a different cost function and does not minimize the negative entropy of $\boldsymbol{u}$ any more. We expect that AdaBoost will have a more uniform distribution of $\boldsymbol{u}$. We run AdaBoost and Arc-Gv with decision stumps on two datasets *breast-cancer* and *australian*.[‖] Fig. 3.1 displays the results. AdaBoost indeed has a small negative entropy of $\boldsymbol{u}$ in both experiments, which agrees with our prediction.

It is evident now that AdaBoost controls the variance of margins by regularizing the Shannon entropy of the corresponding dual variable $\boldsymbol{u}$. For on-line learning algorithms there are two main families of regularization strategies: entropy regularization and regularization using squared Euclidean distance. A question naturally arises here is: What happens if we use squared Euclidean distance to replace the entropy in the dual of AdaBoost (3.16)? In other words, *can we directly minimize the variance of the dual variable $\boldsymbol{u}$ to achieve the purpose of controlling the variance of margins*? We answer this question by having a look at the convex loss functions for classification.

[‖] All datasets used in this work are available at [11] unless otherwise specified.

Fig. 3.2 plots three popular convex loss functions.

It is shown in [2] that, as the data size increases, practically all popular convex loss functions are Bayes-consistent, although convergence rates and other measures of consistency may vary. In the context of boosting, AdaBoost, LogitBoost and soft-margin LPBoost use exponential loss, logistic loss and hinge loss respectively. Here we are interested in the squared hinge loss as it is similar to the hinge loss (no penalty for the positive margins) but differentiable. LogitBoost will be discussed in the next section. As mentioned, in theory, there is no particular reason to prefer hinge loss to squared hinge loss. Now if squared hinge loss is adopted, the cost function of soft-margin LPBoost (3.22) becomes

$$\max_{\boldsymbol{w},\text{margin},\boldsymbol{\xi}} \text{margin} - D\sum_{i=1}^{M} \xi_i^2,$$

and the constraints remain the same as in (3.22). Its dual is easily derived**

$$\begin{aligned} \min_{r,\boldsymbol{u}} \; & r + \tfrac{1}{4D^2}\textstyle\sum_{i=1}^{M} u_i^2 \\ \text{s.t.} \; & \textstyle\sum_{i=1}^{M} u_i y_i H_{ij} \le r \; (\forall j = 1, \cdots, N), \\ & \boldsymbol{u} \succcurlyeq 0, \mathbf{1}^\top \boldsymbol{u} = 1. \end{aligned} \tag{3.28}$$

We can view the above optimization problem as variance regularized LPBoost. In short, to minimize the variance of the dual variable $\boldsymbol{u}$ for controlling the margin's variance, one can simply replace soft-margin LPBoost's hinge loss with the squared hinge loss. Both the primal and dual problems are quadratic programs (QP) and hence can be efficiently solved using off-the-shelf QP solvers like MOSEK [56] or CPLEX [39].

Actually we can generalize the hinge loss into

$$(\max\{0, 1 - yF(\boldsymbol{x})\})^p.$$

When $p \ge 1$, the loss is convex. $p = 1$ is the hinge loss and $p = 2$ is the squared hinge loss. If we use a generalized hinge loss ($p > 1$) for boosting, we end up with a regularized LPBoost which has the format:

$$\min_{r,\boldsymbol{u}} \; r + D'\textstyle\sum_{i=1}^{M} u_i^q, \tag{3.29}$$

subject to the same constraints as in (3.28). Here D' is a constant determined by the primal parameter D and p. p and q are dual to each other by $\frac{1}{p} + \frac{1}{q} = 1$. It is interesting that (3.29) can also be seen as entropy regularized LPBoost; more precisely, Tsallis entropy [86] regularized LPBoost.

**The primal constraint $\boldsymbol{\xi} \succcurlyeq 0$ can be dropped because it is implicitly enforced.

Definition 3.3.2. *(Tsallis entropy) Tsallis entropy is a generalization of the Shannon entropy, defined as*

$$S_q(\boldsymbol{u}) = \frac{1 - \sum_i u_i^q}{q-1}, \quad (\boldsymbol{u} \succcurlyeq 0, \mathbf{1}^\top \boldsymbol{u} = \mathbf{1}). \tag{3.30}$$

where q is a real number. In the limit as $q \to 1$, we have $u_i^{q-1} = \exp((q-1)\log u_i) \simeq 1 + (q-1)\log u_i$. So $S_1 = -\sum_i u_i \log u_i$, which is Shannon entropy.

Tsallis entropy [86] can also be viewed as a q-deformation of Shannon entropy because $S_q(\boldsymbol{u}) = -\sum_i u_i \log_q u_i$ where $\log_q(u) = \frac{u^{1-q}-1}{1-q}$ is the q-logarithm. Clearly $\log_q(u) \to \log(u)$ when $q \to 1$.

In summary, we conclude that although the primal problems of boosting with different loss functions seem dissimilar, their corresponding dual problems share the same formulation. Most of them can be interpreted as entropy regularized LPBoost. Table 3.1 summarizes the result. The analysis of LogitBoost will be presented in the next section.

Table 3.1: Dual problems of boosting algorithms are entropy regularized LPBoost.

algorithm	loss in primal	entropy regularized LPBoost in dual
AdaBoost	exponential loss	Shannon entropy
LogitBoost	logistic loss	binary relative entropy
soft-margin $\ell_p (p > 1)$ LPBoost	generalized hinge loss	Tsallis entropy

3.3.4 Lagrange dual of LogitBoost

Thus far, we have discussed AdaBoost and its relation to LPBoost. In this section, we consider LogitBoost [29] from its dual.

Theorem 3.2. *The dual of LogitBoost is a binary relative entropy maximization problem, which writes*

$$\begin{aligned} \max_{r,\boldsymbol{u}} \ & \frac{r}{T} - \textstyle\sum_{i=1}^M [(-u_i)\log(-u_i) + (1+u_i)\log(1+u_i)] \\ \text{s.t.} \ & \textstyle\sum_{i=1}^M u_i y_i H_{ij} \geq r \ (\forall j = 1, \cdots, N). \end{aligned} \tag{3.31}$$

We can also rewrite it into an equivalent form:

$$\min_{r,\boldsymbol{u}} \ r + T\textstyle\sum_{i=1}^M [u_i \log u_i + (1-u_i)\log(1-u_i)]$$

$$\text{s.t. } \sum_{i=1}^{M} u_i y_i H_{ij} \leq r \ (\forall j = 1, \cdots, N). \tag{3.32}$$

The proof follows the fact that the conjugate of the logistic loss function $\text{logit}(x) = \log(1 + \exp -x)$ is

$$\text{logit}^*(u) = \begin{cases} (-u)\log(-u) + (1+u)\log(1+u), 0 \geq u \geq -1; \\ \infty, \qquad\qquad\qquad\qquad \text{otherwise,} \end{cases}$$

with $0 \log 0 = 0$. $\text{logit}^*(u)$ is a convex function in its domain. The corresponding primal is

$$\begin{aligned} \min_{\boldsymbol{w}} \ & \textstyle\sum_{i=1}^{M} \text{logit}(z_i) \\ \text{s.t. } & z_i = y_i H_i \boldsymbol{w}, \ (\forall i = 1, \cdots, M), \\ & \boldsymbol{w} \succcurlyeq 0, 1^\top \boldsymbol{w} = \tfrac{1}{\mathrm{T}}. \end{aligned} \tag{3.33}$$

In (3.32), the dual variable $\boldsymbol{u}$ has a constraint $1 \succcurlyeq \boldsymbol{u} \succcurlyeq 0$, which is automatically enforced by the logarithmic function. Another difference of (3.32) from duals of AdaBoost and LPBoost, *etc.*, is that $\boldsymbol{u}$ does not need to be normalized. In other words, in LogitBoost the weight associated with each training sample is not necessarily a distribution. As in (3.25) for AdaBoost, we can also relate a dual optimal point $\boldsymbol{u}^\star$ and a primal optimal point $\boldsymbol{w}^\star$ (between (3.32) and (3.33)) by

$$u_i^\star = \frac{\exp -z_i^\star}{1 + \exp -z_i^\star}, \ \forall i = 1, \cdots M. \tag{3.34}$$

So the margin of $\boldsymbol{x}_i$ is solely determined by $u_i^\star$: $z_i^\star = \log \frac{1-u_i^\star}{u_i^\star}$, $\forall i$. For a positive margin ($\boldsymbol{x}_i$ is correctly classified), we must have $u_i^\star < 0.5$.

Similarly, we can also use CG to solve LogitBoost. As shown in Algorithm 3 in the case of AdaBoost, the only modification is to solve a different dual problem (here we need to solve (3.32)).

3.3.5 AdaBoost approximately maximizes the average margin and minimizes the margin variance

Before we present our main result, a lemma is needed.

Lemma 3.3. *The margin of AdaBoost follows the Gaussian distribution. In general, the larger the number of weak classifiers, the more closely the margin follows the form of Gaussian under the assumption that selected weak classifiers are uncorrelated.*

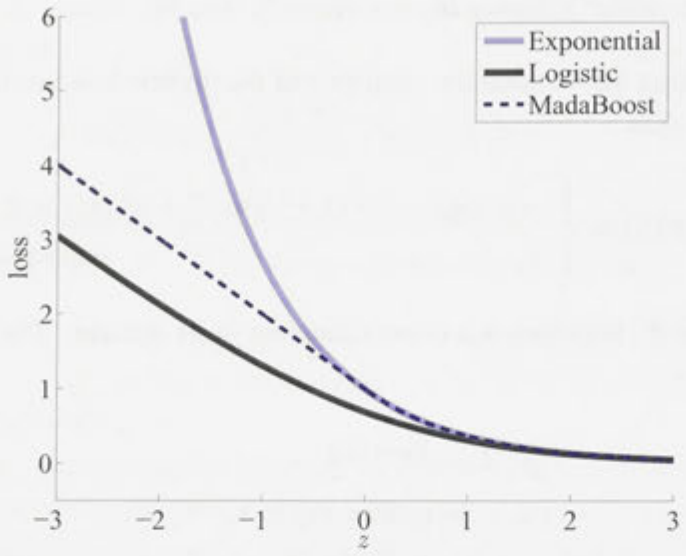


Figure 3.2: Various loss functions used in classification. Exponential: $\exp -s$; logistic: $\log(1+\exp -s)$; squared hinge: $(\max\{0, 1-s\})^2$. Here $s = yF(\boldsymbol{x})$.

Proof: The central limit theorem [40] states that the sum of a set of i.i.d. random variables x_i, $(i = 1 \cdots N)$ is approximately distributed following a Gaussian distribution if the random variables have a finite variance.

Note that the central limit theorem applies when each variable x_i has an *arbitrary* probability distribution $\mathcal{Q}_i$ as long as the variance of $\mathcal{Q}_i$ is finite.

As mentioned, the normalized margin of AdaBoost for i-th example is defined as

$$\text{margin}_i = (y_i \sum_{j=1}^{N} h_j(\boldsymbol{x}_i) w_j) / \mathbf{1}^\top \boldsymbol{w} = -\mathbf{z_i} / \mathbf{1}^\top \boldsymbol{w}. \tag{3.35}$$

In the following analysis, we ignore the normalization term $\mathbf{1}^\top \boldsymbol{w}$ because it does not have any impact on the margin's distribution. Hence the margin ϱ is the sum of N variables $\hat{w}_j$ with $\hat{w}_j = y_i h_j(\boldsymbol{x}_i) w_j$. It is easy to see that each $\hat{w}_j$ follows a discrete distribution, with binary values either w_j or $-w_j$. Therefore w_j must have a finite variance. Using the central limit theorem, we know that the distribution of ϱ_i is a Gaussian.

A condition of the central limit theorem is that the N variables must be independent. In our case, it is well known that usually AdaBoost selects independent weak classifiers such that each weak classifier makes different errors on the training dataset [55]. Therefore, w_j is roughly uncorrelated from each other. More diverse weak classifiers will make the selected weak classifiers less correlated. □

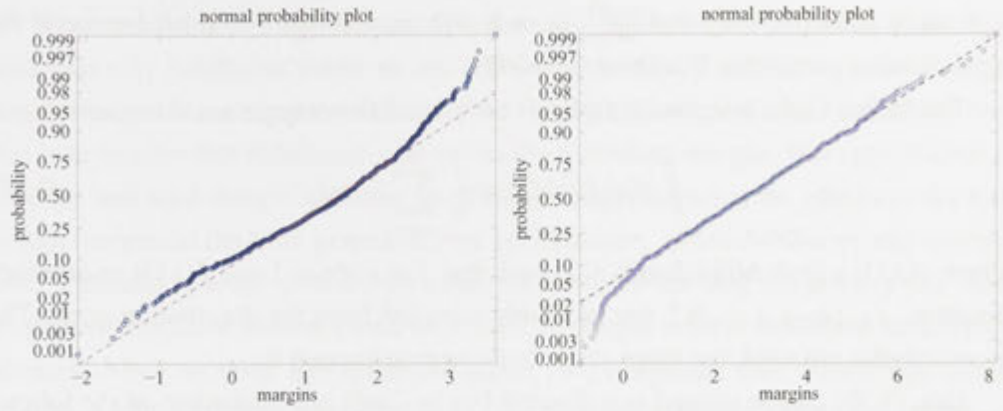


Figure 3.3: Gaussianity test for the margin distribution with 50 and 1100 weak classifiers, respectively. A Gaussian distribution will form a straight line. The dataset used is *australian*.

Here we give some empirical evidence for approximate Gaussianity. The normal (Gaussian) probability plot is used to visually assess whether the data follow a Gaussian distribution. If the data are Gaussian the plot forms a straight line. Other distribution types introduce curvature in the plot. We run AdaBoost with decision stumps on the dataset *australian*. Fig. 3.3 shows two plots of the margins with 50 and 1100 weak classifiers, respectively. We see that with 50 weak classifiers, the margin distribution can be reasonably approximated by a Gaussian; with 1100 classifiers, the distribution is very close to a Gaussian. The kurtosis of a 1D data provides a numerical evaluation of the Gaussianity. We know that the kurtosis of a Gaussian distribution is zero and almost all the other distributions have non-zero kurtosis. In our experiment, the kurtosis is -0.056 for the case with 50 weak classifiers and -0.34 for 1100 classifiers. Both are close to zero, which indicates AdaBoost's margin distribution can be well approximated by Gaussian.

Theorem 3.4. *AdaBoost maximizes the average margin and at the same time minimizes the variance of the margin distribution under the assumption that the margin follows a Gaussian distribution.*

Proof: From (3.5) and (3.35), the cost function that AdaBoost minimizes is

$$f_{\text{ab}}(\boldsymbol{w}) = \log\left(\sum_{i=1}^{M} \exp -\frac{\varrho_i}{T}\right). \tag{3.36}$$

As proved in Lemma 3.3, ϱ_i follows a Gaussian

$$\mathcal{G}(\varrho; \bar{\varrho}, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp -\frac{(\varrho - \bar{\varrho})^2}{2\sigma^2},$$

with mean $\bar{\varrho}$, variance σ^2; and $\sum_{i=1}^{M} \varrho_i = 1$. We assume that the optimal value of the regularization parameter T is known a priori.

The Monte Carlo integration method can be used to compute a continuous integral

$$\int g(x)f(x)dx \simeq \frac{1}{K}\sum_{k=1}^{K} f(x_k), \tag{3.37}$$

where $g(x)$ is a probability distribution such that $\int g(x)dx = 1$ and $f(x)$ is an arbitrary function. x_k, $(k = 1 \cdots K)$, are randomly sampled from the distribution $g(x)$. The more samples are used, the more accurate the approximation is.

Equ. (3.36) can be viewed as a discrete Monte Carlo approximation of the following integral (we omit a constant term $\log M$, which is irrelevant to the analysis):

$$\begin{aligned}
&\hat{f_{\text{ab}}}(\boldsymbol{w}) \\
&= \log \int_{\varrho_1}^{\varrho_2} \mathcal{G}(\varrho; \bar{\varrho}, \sigma) \exp\left(-\frac{\varrho}{T}\right) d\varrho \\
&= \log \int_{\varrho_1}^{\varrho_2} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\varrho-\bar{\varrho})^2}{2\sigma^2} - \frac{\varrho}{T}\right) d\varrho \\
&= \log \left[\tfrac{1}{2} \exp\left(-\frac{\bar{\varrho}}{T} + \frac{\sigma^2}{2T^2}\right) \operatorname{erf}\left(\frac{\varrho-\bar{\varrho}}{\sqrt{2}\sigma} + \frac{\sigma}{\sqrt{2}T}\right)\Bigg|_{\varrho_1}^{\varrho_2}\right] \\
&= -\log 2 - \frac{\bar{\varrho}}{T} + \frac{\sigma^2}{2T^2} + \log \left[\operatorname{erf}\left(\frac{\varrho-\bar{\varrho}}{\sqrt{2}\sigma} + \frac{\sigma}{\sqrt{2}T}\right)\Bigg|_{\varrho_1}^{\varrho_2}\right],
\end{aligned} \tag{3.38}$$

where $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp -s^2 ds$ is the Gauss error function. The integral range is $[\varrho_1, \varrho_2]$. With no explicit knowledge about the integration range, we may roughly calculate the integral from $-\infty$ to $+\infty$. Then the last term in (3.38) is $\log 2$ and the result is analytical and simple

$$\hat{f_{\text{ab}}}(\boldsymbol{w}) = -\frac{\bar{\varrho}}{T} + \frac{1}{2}\frac{\sigma^2}{T^2}. \tag{3.39}$$

This is a reasonable approximation because Gaussian distributions drop off quickly (Gaussian is not considered a heavy-tailed distribution).

Consequently, AdaBoost approximately maximizes the cost function

$$-\hat{f_{\text{ab}}}(\boldsymbol{w}) = \frac{\bar{\varrho}}{T} - \frac{1}{2}\frac{\sigma^2}{T^2}. \tag{3.40}$$

This cost function has a clear and simple interpretation: The first term $\bar{\varrho}/T$ is the unnormalized average margin and the second term σ^2/T^2 is the unnormalized margin variance. So AdaBoost maximizes the unnormalized average margin and also takes minimizing the unnormalized margin variance into account. This way a better *margin distribution* can be obtained. □

Theorem 3.4 is an important result in the sense that it tries to contribute to the open question why AdaBoost works so well. The margin theory in [77] says that the minimum margin is an important indicator for having optimal test error, and much work tends to believe that AdaBoost maximizes the minimum margin. But experiments on Arc-Gv and hard-margin LPBoost [8, 65] empirically question the effects of the minimum margin on the final generalization performance. Since AdaBoost was invented over a decade ago, for such a long time, the machine learning community has failed to present a definite answer about what kind of margin criteria AdaBoost actually optimizes. More recently, Reyzin and Schapire [65] conjecture that the average margin may be a better criterion for optimizing. We have theoretically shown that AdaBoost optimizes the entire margin distribution by maximizing the mean and minimizing the variance of the margin distribution.

We notice that when $T \to 0$, Theorem 3.4 becomes invalid because the Monte Carlo integration cannot approximate the cost function of AdaBoost (3.36) well. In practice, T cannot approach zero arbitrarily in AdaBoost.

One may suspect that Theorem 3.4 contradicts the observation of similarity between LPBoost and AdaBoost as shown in Section 3.3.2. LPBoost maximizes the minimum margin and the dual of AdaBoost is merely an entropy regularized LPBoost. At first glance, the dual variable r in (3.16), (3.21), and (3.23) should have the same meaning, *i.e.*, maximum edge, which in turn corresponds to the minimum margin in the primal. Why average margin? To answer this question, let us again take a look at the optimality conditions. Let us denote the optimal values of (3.16) $r^\star$ and $\boldsymbol{u}^\star$. At convergence, we have $\frac{1}{T}(-r^\star + T\sum_{i=1}^{M} u_i^\star \log u_i^\star) = \mathrm{Opt}^\star_{(3.11)} = \mathrm{Opt}^\star_{(3.4)} = \mathrm{Opt}^\star_{(3.5)}$. Hence, we have

$$r^\star = T\sum_{i=1}^{M} u^\star \log u^\star - T\log\left(\sum_{i=1}^{M}\exp -\frac{\mathrm{margin}_i^\star}{T}\right),$$

where $\mathrm{margin}_i^\star$ is the normalized margin for $\boldsymbol{x}_i$. Clearly this is very different from the optimality conditions of LPBoost, which shows that $r^\star$ is the minimum margin. Only when $T \to 0$, the above relationship reduces to $r^\star = \min_i\{\mathrm{margin}_i^\star\}$—same as the case of LPBoost.

3.3.6 AdaBoost-QP: Direct optimization of the margin mean and variance using quadratic programming

The above analysis suggests that we can directly optimize the cost function (3.40). In this section we show that (3.40) is a convex programming (more precisely, quadratic

programming, QP) problem in the variable $\boldsymbol{w}$ if we know all the base classifiers and hence it can be efficiently solved. Next we formulate the QP problem in detail. We call the proposed algorithm AdaBoost-QP.[††]

In kernel methods like SVMs, the original space $\mathcal{X}$ is mapped to a feature space $\mathcal{F}$. The mapping function $\Phi(\cdot)$ is not explicitly computable. It is shown in [61] that in boosting, one can think of the mapping function $\Phi(\cdot)$ being *explicitly* known:

$$\Phi(\boldsymbol{x}) : \boldsymbol{x} \mapsto [h_1(\boldsymbol{x}), \cdots, h_N(\boldsymbol{x})]^\top, \tag{3.41}$$

using the weak classifiers. Therefore, any weak classifier set $\mathcal{H}$ spans a feature space $\mathcal{F}$. We can design an algorithm that optimizes (3.40):

$$\min_{\boldsymbol{w}} \ \tfrac{1}{2}\boldsymbol{w}^\top A\boldsymbol{w} - T\boldsymbol{b}^\top\boldsymbol{w}, \quad \text{s.t. } \boldsymbol{w} \succcurlyeq 0, \mathbf{1}^\top\boldsymbol{w} = 1, \tag{3.42}$$

where $\boldsymbol{b} = \frac{1}{M}\sum_{i=1}^{M} y_i H_i^\top = \frac{1}{M}\sum_{i=1}^{M} y_i \Phi(\boldsymbol{x}_i)$, and $A = \frac{1}{M}\sum_{i=1}^{M}(y_i H_i^\top - \boldsymbol{b})(y_i H_i^\top - \boldsymbol{b})^\top = \frac{1}{M}\sum_{i=1}^{M}(y_i\Phi(\boldsymbol{x}_i) - \boldsymbol{b})(y_i\Phi(\boldsymbol{x}_i) - \boldsymbol{b})^\top$.[‡‡] Clearly A must be positive semidefinite and this is a standard convex QP problem. The non-negativeness constraint $\boldsymbol{w} \succcurlyeq 0$ introduces sparsity as in SVMs. Without this constraint, the above QP can be analytically solved using eigenvalue decomposition—the largest eigenvector is the solution. Usually all entries of this solution would be active (non-zero values).

In the kernel space,

$$\boldsymbol{b}^\top\boldsymbol{w} = \tfrac{1}{M}\left(\sum\nolimits_{y_i=1}\Phi(\boldsymbol{x}_i) - \sum\nolimits_{y_i=-1}\Phi(\boldsymbol{x}_i)\right)^\top \boldsymbol{w}$$

can be viewed as the projected ℓ_1 norm distance between two classes because typically this value is positive, assuming that each class has the same number of examples. The matrix A *approximately* plays a role as the total scatter matrix in kernel linear discriminant analysis (LDA). Note that AdaBoost does not take the number of examples in each class into consideration when it models the problem. In contrast, LDA (kernel LDA) takes training example number into consideration. This may explain why an LDA post-processing on AdaBoost gives a better classification performance on face detection [99], which is a highly imbalanced classification problem. This observation of similarity between AdaBoost and kernel LDA may inspire new algorithms. We are also interested in developing a CG based algorithm for iteratively generating weak classifiers.

[††] In [62], the authors proposed QP_{reg}-AdaBoost for soft-margin AdaBoost learning, which is inspired by SVMs. Their QP_{reg}-AdaBoost is completely different from ours.

[‡‡] To show the connection of AdaBoost-QP with kernel methods, we have written $\Phi(\boldsymbol{x}_i) = H_i^\top$.

3.3.7 AdaBoost-CG: Totally corrective AdaBoost using column-generation

The number of possible weak classifiers may be infinitely large. In this case it may be infeasible to solve the optimization *exactly*. AdaBoost works on the primal problem directly by switching between the estimating weak classifiers and computing optimal weights in a coordinate descent way. However, similar to LPBoost, we also can employ the column-generation method to solve the optimization problem.

In our case, instead of solving the optimization of AdaBoost directly, one computes the most violated constraint in (3.16) iteratively for the current solution and adds this constraint to the optimization problem. In theory, any column that violates dual feasibility can be added. To do so, we need to solve the following subproblem:

$$h'(\cdot) = \underset{h(\cdot)}{\operatorname{argmax}} \ \sum_{i=1}^{M} u_i y_i h(\boldsymbol{x}_i). \tag{3.43}$$

This strategy is exactly the same as the one that stage-wise AdaBoost and LPBoost use for generating the best weak classifier: that is, to find the weak classifier that produces minimum weighted training error. Putting all the above analysis together, we summarize our AdaBoost-CG in Algorithm 3.

The CG optimization (Algorithm 3) is so general that it can be applied to all the boosting algorithms considered in this chapter by solving the corresponding dual. The convergence follows general CG algorithms, which is easy to establish. When a new $h'(\cdot)$ that violates dual feasibility is added, the new optimal value of the dual problem (maximization) would decrease. Accordingly, the optimal value of its primal problem decreases too because they have the same optimal value due to a zero duality gap. Moreover the primal cost function is convex, therefore eventually it converges to the global minimum. A comment on the last step of Algorithm 3 is that we can get the value of $\boldsymbol{w}$ easily. Primal-dual interior-point (PD-IP) methods work on the primal and dual problems simultaneously and therefore both primal and dual variables are available after convergence. We use MOSEK [56], which implements PD-IP methods. The primal variable $\boldsymbol{w}$ is obtained *for free* when solving the dual problem (3.16).

The dual subproblem we need to solve has one constraint added at each iteration. Hence after many iterations solving the dual problem could become intractable in theory. In practice, AdaBoost-CG converges quickly on our tested datasets. As pointed out in [84], usually only a small number of the added constraints are active and those inactive ones may be removed. This strategy prevents the dual problem from growing too large.

AdaBoost-CG is totally corrective in the sense that the coefficients of all weak classifiers are updated at each iteration. In [83], an additional correction procedure is inserted to AdaBoost's weak classifier selection cycle for achieving totally-correction. The inserted correction procedure aggressively reduces the upper bound of the training error. Like AdaBoost, it works in the primal. In contrast, our algorithm optimizes the regularized loss function directly and works mainly in the dual space.

The following diagram summarizes the relationships that we have derived on AdaBoost.

Figure 3.4: The diagram summarizes the relations between the proposed and some existing boosting algorithms mentioned in this chapter.

3.4 Experiments

In this section we provide experimental results to verify the presented theory. We have mainly used decision stumps as weak classifiers due to its simplicity and well-controlled complexity. In some cases, we have also used one of the simplest linear classifiers, LDA, as weak classifiers. To avoid the singularity problem when solving LDA, we add a scaled identity matrix $10^{-4}\mathbf{I}$ to the within-class matrix. For the CG optimization framework, we have confined ourself to AdaBoost-CG, although the technique is general and applicable for optimizing other boosting algorithms.

3.4.1 AdaBoost-QP

We compare AdaBoost-QP against AdaBoost. We have used 14 benchmark datasets [11]. All the other datasets, except *mushrooms*, *svmguide1*, *svmguide3* and *w1a*, have been scaled to $[-1, 1]$. We randomly split each dataset into training, cross-validation and test sets at a ratio of $70 : 15 : 15$. Table 3.2 provides a description of the datasets we have used in the experiments.

The stopping criterion of AdaBoost is determined by cross-validation on $\{600, 800, 1000, 1200, 1500\}$ rounds of boosting. For AdaBoost-QP, the best value for the parameter T is chosen from $\{\frac{1}{10}, \frac{1}{20}, \frac{1}{30}, \frac{1}{40}, \frac{1}{50}, \frac{1}{100}, \frac{1}{200}, \frac{1}{500}\}$ by cross-validation. In this

dataset	# examples	# features	dataset	# examples	# features
australian	690	14	**liver-disorders**	345	6
breast-cancer	683	10	**mushrooms**	8124	112
diabetes	768	8	**sonar**	208	60
fourclass	862	2	**splice**	1000	60
german-numer	1000	24	**svmguide1**	3089	4
heart	270	13	**svmguide3**	1243	22
ionosphere	351	34	**w1a**	2477	300

Table 3.2: Description of the datasets. Except *mushrooms*, *svmguide1*, *svmguide3* and *w1a*, all the other datasets have been scaled to $[-1, 1]$.

experiment, decision stumps are used as the weak classifier, such that the complexity of the base classifiers is well controlled.

AdaBoost-QP must access all weak classifiers *a priori*. Here we run AdaBoost-QP on the 1500 weak classifiers generated by AdaBoost. Clearly this number of hypotheses may not be optimal. Theoretically the larger the size of the weak classifier pool is, the better results AdaBoost-QP may produce. Table 3.3 reports the results. The experiments show that among these 14 datasets, AdaBoost-QP outperforms AdaBoost on 9 datasets in terms of generalization error. On *mushrooms*, both perform very well. On the other 4 datasets, AdaBoost is better.

We have also computed the normalized version of the cost function value of (3.40). In most cases AdaBoost-QP has a larger value. This is not surprising since AdaBoost-QP directly maximizes (3.40) while AdaBoost approximately maximizes it. Furthermore, the normalized loss function value is close to the normalized average margin because the margin variances for most datasets are very small compared with their means.

We also compute the largest minimum margin and average margin on each dataset. On all the datasets AdaBoost has a larger minimum margin than AdaBoost-QP. This confirms that the minimum margin is not crucial for the generalization error. On the other hand, the average margin produced by AdaBoost-QP, which is the first term of the cost function (3.40), is consistently larger than the one obtained by AdaBoost. Indirectly, we have shown that a better overall margin distribution is more important than the largest minimum margin. In Fig. 3.5 we plot cumulative margins for AdaBoost-QP and AdaBoost on the breast-cancer dataset with decision stumps. We can see while Arc-Gv has a largest minimum margin, it has a worst margins distribution overall. If we examine the average margins, AdaBoost-QP is the largest; AdaBoost is the sec-

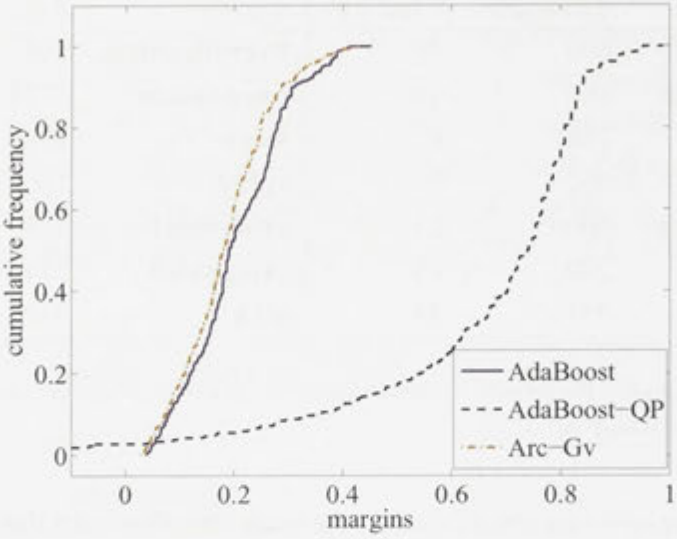


Figure 3.5: Cumulative margins for AdaBoost, AdaBoost-QP and Arc-Gv for the breast cancer dataset using decision stumps. Overall, the margin distribution of AdaBoost-QP is the best and it has a smallest test error. AdaBoost and Arc-Gv run 600 rounds of boosting. Test error for AdaBoost, AdaBoost-QP and Arc-Gv is $0.029, 0.027, 0.058$ respectively.

ond and Arc-Gv is least. Clearly a better overall distribution does lead to a smaller generalization error. When Arc-Gv and AdaBoost run for more rounds, their margin distributions seem to converge. That is what we see in Fig. 3.5. These results agree well with our theoretical analysis (Theorem 3.4). Another observation is that, to achieve the same performance, AdaBoost-QP tends to use fewer weak classifiers than AdaBoost does.

We have also tested AdaBoost-QP on full sets of weak classifiers because the number of possible decision stumps is finite (less than (number of features -1) $\times$ (number of examples)). Table 3.4 reports the test error of AdaBoost-QP on some small datasets. As expected, in most cases the test error is slightly better than the results using 1500 decision stumps in Table 3.3; and no significant difference is observed. This verifies the capability of AdaBoost-QP for selecting and combining relevant weak classifiers.

3.4.2 AdaBoost-CG

We run AdaBoost and AdaBoost-CG with decision stumps on the datasets of [11]. 70% of examples are used for training; 15% are used for test and the other 15% are not used because we do not do cross-validation here. The convergence threshold for

AdaBoost-CG (ε in Algorithm 3) is set to 10^{-5}. Another important parameter to tune is the regularization parameter T. For the first experiment, we have set it to $1/\mathbf{1}^\top \boldsymbol{w}$ where $\boldsymbol{w}$ is obtained by running AdaBoost on the same data for 1000 iterations. Also for fair comparison, we have deliberately forced AdaBoost-CG to run 1000 iterations even if the stopping criterion is met. Both test and training results for AdaBoost and AdaBoost-CG are reported in Table 3.5 for a maximum number of iterations of 100, 500 and 1000.

As expected, in terms of test error no algorithm statistically outperforms the other one, since they optimize the same cost function. As we can see, AdaBoost does slightly better on 6 datasets. AdaBoost-CG outperforms AdaBoost on 7 datasets and on *svmguide1*, both algorithms perform almost identically. Therefore, empirically we conclude that in terms of generalization capability, AdaBoost-CG is the same as the standard AdaBoost.

However, in terms of training error and convergence speed of the training procedure, there is significant difference between these two algorithms. Looking at the right part of Table 3.5, we see that the training error of AdaBoost-CG is consistently better or no worse than AdaBoost on *all* tested datasets. We have the following conclusions.

- The convergence speed of AdaBoost-CG is faster than AdaBoost and in many cases, better training error can be achieved. This is because AdaBoost's coordinate descent nature is slow while AdaBoost-CG is *totally corrective*[§§]. This also means that with AdaBoost-CG, we can use fewer weak classifiers to build a good strong classifier. This is desirable for real-time applications like face detection [91], in which the testing speed is critical.

- Our experiments confirm that a smaller training error does not necessarily lead to a smaller test error. This has been studied extensively in statistical learning theory. It is observed that AdaBoost sometimes suffers from overfitting, and minimizing the exponential cost function of the margins does not solely determine test error.

In the second experiment, we run cross-validation to select the best value for the regularization parameter T, as in Section 3.4.1. Table 3.6 reports the test errors on a subset of the datasets. Slightly better results are obtained compared with the results in Table 3.5, which uses T determined by AdaBoost.

We also use LDA as weak classifiers to compare the classification performance of AdaBoost and AdaBoost-CG. The parameter T of AdaBoost-CG is determined by

[§§]Like LPBoost, at each iteration AdaBoost-CG updates the previous weak classifier weights $\boldsymbol{w}$.

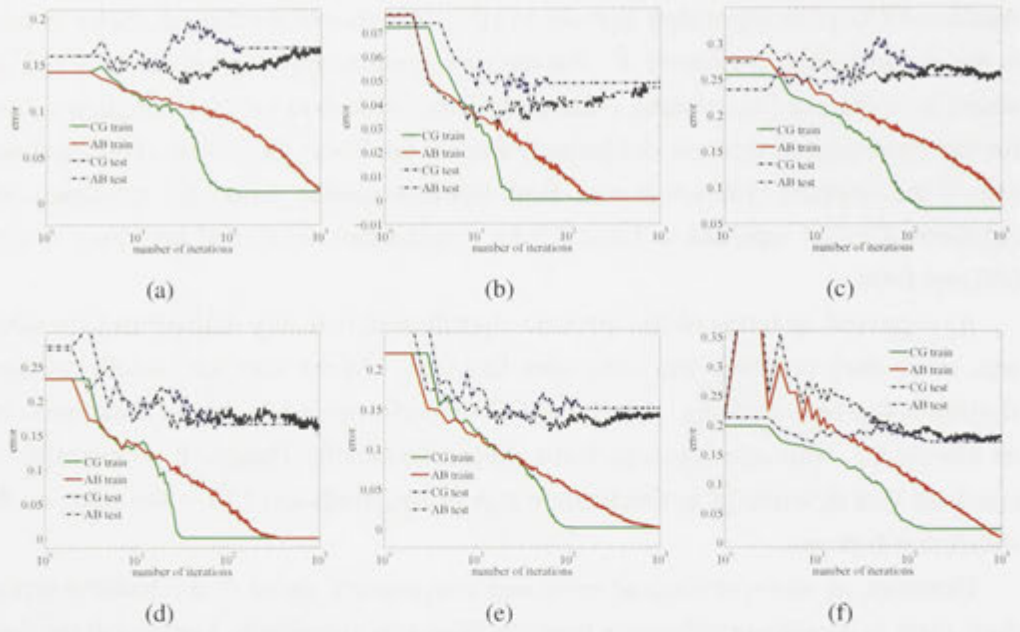


Figure 3.6: Test error and training error of AdaBoost, AdaBoost-CG for *australian* (a), *breast-cancer* (b), *diabetes* (c), *heart* (d), *spline* (e) and *svmguide3* (f) datasets. These convergence curves correspond to the results in Table 3.5. The x-axis is on a logarithmic scale for easier comparison.

cross-validation from $\{\frac{1}{2}, \frac{1}{5}, \frac{1}{8}, \frac{1}{10}, \frac{1}{12}, \frac{1}{15}, \frac{1}{20}, \frac{1}{30}, \frac{1}{40}, \frac{1}{50}, \frac{1}{70}, \frac{1}{90}, \frac{1}{100}, \frac{1}{120}, \frac{1}{150}\}$. For AdaBoost the smallest test error from 100, 500 and 1000 runs is reported. We show the results in Table 3.7. As we can see, the test error is slightly better than with decision stumps for both AdaBoost and AdaBoost-CG. Again, AdaBoost and AdaBoost-CG's performances are very similar.

In order to show that statistically there is no difference between AdaBoost-CG and AdaBoost, the McNemar test [19] with a significance level of 0.05 is conducted. McNemar's test is based on a χ^2 test [19]. If the quantity of the χ^2 test is not greater than $\chi^2_{1,0.95} = 3.841459$, we can think that the two tested classifiers have *no statistical difference* in terms of classification capability. On the 8 datasets with decision stumps and LDA (Tables 3.6 and 3.7), in all cases (5 runs per dataset), the results of χ^2 test are smaller than $\chi^2_{1,0.95}$. Consequently, we can conclude that indeed AdaBoost-CG performs very similarly to AdaBoost for classification.

To examine the effect of parameter T, we run more experiments with various T on the *banana* dataset (2D artificial data) that was used in [62]. We still use decision stumps. The maximum iteration is set to 400. All runs stop earlier than 100 iterations. Table 3.8 reports the results. Indeed, the training error depends on T. T also has

influence on the convergence speed. But, in a wide range of T, the test error does not change significantly. We do not have a sophisticated technique to tune T. As mentioned, the sum of $\boldsymbol{w}$ from a run of AdaBoost can serve as a heuristic.

Now let us take a close look at the convergence behavior of AdaBoost-CG. Fig. 3.6 shows the test and training error of AdaBoost and AdaBoost-CG for 6 datasets. We see that AdaBoost-CG converges much faster than AdaBoost in terms of number of iterations. On most tested datasets, AdaBoost-CG is around 10 times faster than AdaBoost. The test errors for these two methods are very close upon convergence. In some datasets such as *australian* and *breast-cancer* we observe overfitting for AdaBoost.

3.5 Discussion and Conclusion

In this work, we have shown that the Lagrange dual problems of AdaBoost, LogitBoost and soft-margin LPBoost with generalized hinge loss are all entropy regularized LPBoost. We both theoretically and empirically demonstrate that the success of AdaBoost relies on maintaining a better margin distribution.

Based on the dual formulation, a general column-generation based optimization framework is proposed. This optimization framework can be applied to solve all the boosting algorithms with various loss functions mentioned in this chapter. Experiments with exponential loss show that the classification performance of AdaBoost-CG is statistically almost identical to the standard stage-wise AdaBoost on real datasets. In fact, since both algorithms optimize the same cost function, we would be surprised to see a significant different in their generalization error.

The main advantage of the proposed algorithms is significantly faster convergence speed.

Compared with the conventional AdaBoost, a drawback of AdaBoost-CG is the introduction of a parameter, same as in LPBoost. While one can argue that AdaBoost implicitly determines this same parameter by selecting how many iterations to run, the stopping criterion is nested and thus efficient to learn. In the case of AdaBoost-CG, it is not clear how to efficiently learn this parameter. Currently, one has to run the training procedure multiple times for cross validation.

With the optimization framework established here, some issues on boosting that are previously unclear may become obvious now. For example, for designing cost-sensitive boosting or boosting on uneven datasets, one can simply modify the primal cost function (3.4) to have a weighted cost function [43]. The training procedure fol-

lows AdaBoost-CG.

To summarize, the convex duality of boosting algorithms presented in this work generalizes the convex duality in LPBoost. We have shown some interesting properties that the derived dual formation possesses. The duality also leads to new efficient learning algorithms. The duality provides useful insights on boosting that may be lacking in existing interpretations [77, 29].

In the future, we want to extend our work to boosting with non-convex loss functions such as BrownBoost [23]. Also it should be straightforward to optimize boosting for regression using column-generation. We are currently exploring the application of AdaBoost-CG to efficient object detection, due to its faster convergence, which is more promising for feature selection [91]. A totally corrective version of AdaBoost in [83] has proved its efficiency on face detection.

Algorithm 3 AdaBoost-CG.

Input: Training set $(\boldsymbol{x}_i, y_i), i = 1 \cdots M$; termination threshold $\varepsilon > 0$; regularization parameter T; (optional) maximum iteration $N_{\max}$.

Initialization:

1. $N = 0$ (no weak classifiers selected);
2. $\boldsymbol{w} = \mathbf{0}$ (all primal coefficients are zeros);
3. $u_i = \frac{1}{M}, i = 1 \cdots M$ (uniform dual weights).

while true **do**

1. Find a new base $h'(\cdot)$ by solving Problem (3.43);
2. Check for optimal solution:
   if $\sum_{i=1}^{M} u_i y_i h'(\boldsymbol{x}_i) < r + \varepsilon$, **then** break (problem solved);
3. Add $h'(\cdot)$ to the restricted master problem, which corresponds to a new constraint in the dual;
4. Solve the dual to obtain updated r and u_i $(i = 1, \cdots, M)$: for AdaBoost, the dual is (3.16);
5. $N = N + 1$ (weak classifier count);
6. (optional) **if** $N \geq N_{\max}$, **then** break (maximum iteration reached).

end

Output:

1. Calculate the primal variable $\boldsymbol{w}$ from the optimality conditions and the last solved dual problem;
2. The learned classifier $F(\boldsymbol{x}) = \sum_{j=1}^{N} w_j h_j(\boldsymbol{x})$.

dataset	algorithm	test error	minimum margin	average margin
australian	AB	0.153 ± 0.034	$\mathbf{-0.012 \pm 0.005}$	0.082 ± 0.006
	QP	$\mathbf{0.13 \pm 0.038}$	-0.227 ± 0.081	$\mathbf{0.18 \pm 0.052}$
b-cancer	AB	0.041 ± 0.013	$\mathbf{0.048 \pm 0.009}$	0.209 ± 0.02
	QP	$\mathbf{0.03 \pm 0.012}$	-0.424 ± 0.250	$\mathbf{0.523 \pm 0.237}$
diabetes	AB	0.270 ± 0.043	$\mathbf{-0.038 \pm 0.007}$	0.055 ± 0.005
	QP	$\mathbf{0.262 \pm 0.047}$	-0.107 ± 0.060	$\mathbf{0.075 \pm 0.031}$
fourclass	AB	$\mathbf{0.088 \pm 0.032}$	$\mathbf{-0.045 \pm 0.012}$	0.084 ± 0.009
	QP	0.095 ± 0.028	-0.211 ± 0.059	$\mathbf{0.128 \pm 0.027}$
g-numer	AB	0.283 ± 0.033	$\mathbf{-0.079 \pm 0.017}$	0.042 ± 0.006
	QP	$\mathbf{0.249 \pm 0.033}$	-0.151 ± 0.058	$\mathbf{0.061 \pm 0.020}$
heart	AB	0.210 ± 0.032	$\mathbf{0.02 \pm 0.008}$	0.104 ± 0.013
	QP	$\mathbf{0.190 \pm 0.058}$	-0.117 ± 0.066	$\mathbf{0.146 \pm 0.059}$
ionosphere	AB	$\mathbf{0.121 \pm 0.044}$	$\mathbf{0.101 \pm 0.010}$	0.165 ± 0.012
	QP	0.139 ± 0.055	-0.035 ± 0.112	$\mathbf{0.184 \pm 0.063}$
liver	AB	0.321 ± 0.040	$\mathbf{-0.012 \pm 0.007}$	0.055 ± 0.005
	QP	$\mathbf{0.314 \pm 0.060}$	-0.107 ± 0.044	$\mathbf{0.079 \pm 0.021}$
mushrooms	AB	$\mathbf{0 \pm 0}$	$\mathbf{0.102 \pm 0.001}$	0.181 ± 0.001
	QP	0.005 ± 0.002	-0.134 ± 0.086	$\mathbf{0.221 \pm 0.084}$
sonar	AB	$\mathbf{0.145 \pm 0.046}$	$\mathbf{0.156 \pm 0.008}$	0.202 ± 0.013
	QP	0.171 ± 0.048	0.056 ± 0.066	$\mathbf{0.220 \pm 0.045}$
splice	AB	0.129 ± 0.025	$\mathbf{-0.009 \pm 0.008}$	0.117 ± 0.009
	QP	$\mathbf{0.106 \pm 0.029}$	-0.21 ± 0.037	$\mathbf{0.189 \pm 0.02}$
svmguide1	AB	$\mathbf{0.035 \pm 0.009}$	$\mathbf{-0.010 \pm 0.008}$	0.157 ± 0.016
	QP	0.040 ± 0.009	-0.439 ± 0.183	$\mathbf{0.445 \pm 0.155}$
svmguide3	AB	0.172 ± 0.023	$\mathbf{-0.011 \pm 0.009}$	0.052 ± 0.005
	QP	$\mathbf{0.167 \pm 0.022}$	-0.113 ± 0.084	$\mathbf{0.085 \pm 0.038}$
w1a	AB	0.041 ± 0.014	$\mathbf{-0.048 \pm 0.010}$	0.084 ± 0.005
	QP	$\mathbf{0.029 \pm 0.009}$	-0.624 ± 0.38	$\mathbf{0.577 \pm 0.363}$

Table 3.3: Test results of AdaBoost (AB) and AdaBoost-QP (QP). All tests are run 10 times. The mean and standard deviation are reported. AdaBoost-QP outperforms AdaBoost on 9 datasets.

dataset	**australian**	**b-cancer**	**fourclass**	**g-numer**	**heart**	**liver**	**mushroom**	**splice**
test error	0.131 ± 0.041	0.03 ± 0.011	0.091 ± 0.02	0.243 ± 0.026	0.188 ± 0.058	0.319 ± 0.05	0.003 ± 0.001	0.097 ± 0.02

Table 3.4: Test results of AdaBoost-QP on full sets of decision stumps. All tests are run 10 times.

dataset	algorithm	test error 100	test error 500	test error 1000	train error 100	train error 500	train error 1000
australian	AB	$\mathbf{0.146 \pm 0.028}$	$\mathbf{0.165 \pm 0.018}$	$\mathbf{0.163 \pm 0.021}$	0.091 ± 0.013	0.039 ± 0.011	0.013 ± 0.009
	CG	0.177 ± 0.025	0.167 ± 0.023	0.167 ± 0.023	$\mathbf{0.013 \pm 0.008}$	$\mathbf{0.011 \pm 0.007}$	$\mathbf{0.011 \pm 0.007}$
b-cancer	AB	$\mathbf{0.041 \pm 0.026}$	$\mathbf{0.045 \pm 0.030}$	$\mathbf{0.047 \pm 0.032}$	0.008 ± 0.006	0 ± 0	0 ± 0
	CG	0.049 ± 0.033	0.049 ± 0.033	0.049 ± 0.033	$\mathbf{0 \pm 0}$	0 ± 0	0 ± 0
diabetes	AB	$\mathbf{0.254 \pm 0.024}$	0.263 ± 0.028	0.257 ± 0.041	0.171 ± 0.012	0.120 ± 0.007	0.082 ± 0.006
	CG	0.270 ± 0.047	$\mathbf{0.254 \pm 0.026}$	$\mathbf{0.254 \pm 0.026}$	$\mathbf{0.083 \pm 0.008}$	$\mathbf{0.070 \pm 0.007}$	$\mathbf{0.070 \pm 0.007}$
fourclass	AB	0.106 ± 0.047	0.097 ± 0.034	0.091 ± 0.031	0.072 ± 0.023	0.053 ± 0.017	0.046 ± 0.017
	CG	$\mathbf{0.082 \pm 0.031}$	$\mathbf{0.082 \pm 0.031}$	$\mathbf{0.082 \pm 0.031}$	$\mathbf{0.042 \pm 0.015}$	$\mathbf{0.042 \pm 0.015}$	$\mathbf{0.042 \pm 0.015}$
g-numer	AB	0.279 ± 0.043	0.288 ± 0.048	0.297 ± 0.051	0.206 ± 0.047	0.167 ± 0.072	0.155 ± 0.082
	CG	$\mathbf{0.269 \pm 0.040}$	$\mathbf{0.262 \pm 0.045}$	$\mathbf{0.262 \pm 0.045}$	$\mathbf{0.142 \pm 0.077}$	$\mathbf{0.142 \pm 0.077}$	$\mathbf{0.142 \pm 0.077}$
heart	AB	0.175 ± 0.073	0.175 ± 0.088	0.165 ± 0.076	0.049 ± 0.022	0 ± 0	0 ± 0
	CG	$\mathbf{0.165 \pm 0.072}$	$\mathbf{0.165 \pm 0.072}$	$\mathbf{0.165 \pm 0.072}$	$\mathbf{0 \pm 0}$	0 ± 0	0 ± 0
ionosphere	AB	$\mathbf{0.092 \pm 0.016}$	$\mathbf{0.104 \pm 0.017}$	$\mathbf{0.100 \pm 0.016}$	0 ± 0	0 ± 0	0 ± 0
	CG	0.131 ± 0.034	0.131 ± 0.034	0.131 ± 0.034	0 ± 0	0 ± 0	0 ± 0
liver	AB	0.288 ± 0.101	$\mathbf{0.265 \pm 0.081}$	$\mathbf{0.281 \pm 0.062}$	0.144 ± 0.018	0.063 ± 0.015	0.020 ± 0.015
	CG	$\mathbf{0.288 \pm 0.084}$	0.288 ± 0.084	0.288 ± 0.084	$\mathbf{0.017 \pm 0.012}$	$\mathbf{0.017 \pm 0.011}$	$\mathbf{0.017 \pm 0.011}$
mushrooms	AB	0 ± 0.001	0 ± 0.001	0 ± 0	0 ± 0	0 ± 0	0 ± 0
	CG	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	0 ± 0	0 ± 0	0 ± 0	0 ± 0
sonar	AB	$\mathbf{0.206 \pm 0.087}$	$\mathbf{0.213 \pm 0.071}$	$\mathbf{0.206 \pm 0.059}$	0 ± 0	0 ± 0	0 ± 0
	CG	0.232 ± 0.053	0.245 ± 0.078	0.245 ± 0.078	0 ± 0	0 ± 0	0 ± 0
splice	AB	$\mathbf{0.129 \pm 0.011}$	$\mathbf{0.143 \pm 0.026}$	$\mathbf{0.143 \pm 0.020}$	0.053 ± 0.003	0.008 ± 0.006	0.001 ± 0.001
	CG	0.161 ± 0.033	0.151 ± 0.023	0.151 ± 0.023	$\mathbf{0.002 \pm 0.002}$	$\mathbf{0.001 \pm 0.002}$	$\mathbf{0.001 \pm 0.002}$
svmguide1	AB	$\mathbf{0.036 \pm 0.012}$	$\mathbf{0.034 \pm 0.008}$	0.037 ± 0.007	0.022 ± 0.002	0.009 ± 0.002	0.002 ± 0.001
	CG	0.037 ± 0.007	0.037 ± 0.007	0.037 ± 0.007	$\mathbf{0.001 \pm 0.001}$	$\mathbf{0 \pm 0.001}$	$\mathbf{0 \pm 0.001}$
svmguide3	AB	0.184 ± 0.037	0.183 ± 0.044	0.182 ± 0.031	0.112 ± 0.009	0.037 ± 0.004	0.009 ± 0.003
	CG	$\mathbf{0.184 \pm 0.026}$	$\mathbf{0.171 \pm 0.023}$	$\mathbf{0.171 \pm 0.023}$	$\mathbf{0.033 \pm 0.012}$	$\mathbf{0.023 \pm 0.016}$	$\mathbf{0.023 \pm 0.016}$
w1a	AB	0.051 ± 0.009	0.038 ± 0.005	0.036 ± 0.004	0.045 ± 0.008	0.028 ± 0.005	0.025 ± 0.005
	CG	$\mathbf{0.018 \pm 0.001}$	$\mathbf{0.018 \pm 0.001}$	$\mathbf{0.018 \pm 0.001}$	$\mathbf{0.010 \pm 0.004}$	$\mathbf{0.010 \pm 0.004}$	$\mathbf{0.010 \pm 0.004}$

Table 3.5: Test and training errors of AdaBoost (AB) and AdaBoost-CG (CG). All tests are run 5 times. The mean and standard deviation are reported. Weak classifiers are decision stumps.

dataset	**australian**	**b-cancer**	**diabetes**	**fourclass**	**heart**	**ionosphere**	**sonar**	**splice**
test error	0.146 ± 0.027	0.033 ± 0.033	0.266 ± 0.036	0.086 ± 0.027	0.17 ± 0.082	0.115 ± 0.024	0.2 ± 0.035	0.135 ± 0.015

Table 3.6: Test error of AdaBoost-CG with decision stumps, using cross-validation to select the optimal T. All tests are run 5 times.

dataset	**australian**	**b-cancer**	**diabetes**	**fourclass**	**heart**	**ionosphere**	**sonar**	**splice**
AB	0.150 ± 0.044	0.029 ± 0.014	0.259 ± 0.021	0.003 ± 0.004	0.16 ± 0.055	0.108 ± 0.060	0.297 ± 0.080	0.215 ± 0.027
CG	0.151 ± 0.053	0.035 ± 0.016	0.249 ± 0.038	0.022 ± 0.015	0.185 ± 0.038	0.104 ± 0.062	0.258 ± 0.085	0.235 ± 0.035

Table 3.7: Test error of AdaBoost (AB) and AdaBoost-CG (CG) with LDA as weak classifiers, using cross-validation to select the optimal T. All tests are run 5 times.

$\frac{1}{\gamma}$	test (stumps)	train (stumps)	test (LDA)	train (LDA)
20	0.298 ± 0.018	0.150 ± 0.019	0.134 ± 0.012	0.032 ± 0.007
40	0.309 ± 0.019	0.101 ± 0.015	0.135 ± 0.008	0.001 ± 0.002
80	0.313 ± 0.019	0.033 ± 0.011	0.136 ± 0.007	0 ± 0

Table 3.8: AdaBoost-CG on *banana* dataset with decision stumps and LDA as weak classifiers. Experiments are run 50 times.

Chapter 4

Boosting through Optimization of Margin Distributions

4.1 Introduction

As introduced in the previous chapters, there are already many empirical evidences suggesting that a good margin distribution is more important than a large minimum margin [35, 64, 97, 16, 65]. Recently, Garg and Roth [33] introduced a margin distribution based complexity measure for learning classifiers and developed margin distribution based generalization bounds. Competitive classification results have been shown by optimizing this bound. Another relevant work is [47], which applies a boosting method to optimize the margin distribution based generalization bound obtained by [79]. Experiments show that the new boosting methods achieve considerable improvements over AdaBoost. The optimization of this new boosting method is based on the AnyBoost framework [53]. Aligned with these attempts, we propose a new boosting-like algorithm through optimization of margin distribution (termed MDBoost). Instead of minimizing a margin distribution based generalization bound, we directly optimize the margin distribution: maximizing the average margin and at the same time minimizing the variance of the margin distribution.

The theoretical justification of the proposed MDBoost is that, approximately, AdaBoost actually maximizes the average margin and minimizes the margin variance.

The main contributions of this chapter are as follows.

1. We propose a new totally corrective boosting algorithm, MDBoost, by optimizing the margin distribution directly. The optimization procedure of MDBoost is based on the idea of column-generation that has been widely used in large-scale linear programming.

2. We empirically demonstrate that MDBoost outperforms AdaBoost on most UCI data sets used in our experiments. The success of MDBoost verifies the conjecture in [65].

3. AdaBoost-CG [81] is also totally corrective in the sense that all the linear coefficients of the weak classifiers are updated during the training. Our MDBoost has similar classification performance compared with AdaBoost-CG, since they approximately optimize the same cost functions. However, MDBoost is *faster* in training because MDBoost solves a quadratic program at each iteration while AdaBoost-CG needs to solve a general convex program.

The rest of the chapter is structured as follows. In Section 4.2 we present the main idea. In Section 4.3.1 the dual of the MDBoost's optimization problem is derived, which enables us to design an LPBoost-like column-generation based boosting algorithm. We provide an experimental comparison of the algorithms on UCI data in Section 4.4, and conclude the work in Section 4.5.

4.2 Algorithm

Let us define the final output strong classifier of boosting algorithms as $F(\boldsymbol{x}) = \sum_{j=1}^{N} w_j h_j(\boldsymbol{x})$ with $w_j > 0, j = 1 \cdots N$. The following theorem serves as the basis of the proposed MDBoost algorithm. The detailed proof can be found in Chapter 3.

Theorem 4.1. *AdaBoost maximizes the unnormalized average margin and simultaneously minimizes the variance of the margin distribution under the assumption that the margin follows a Gaussian distribution.*

Proof: See Chapter 3. □

The key assumption that makes this theorem valid is that the weak learners generated by AdaBoost make independent errors over the training data set. This assumption may not be true in practice, but could be a plausible approximation.

Mathematically, the above theorem can be formulated as:

$$\max_{\boldsymbol{w}} \bar{\rho} - \tfrac{1}{2}\sigma^2, \text{ s.t. } \boldsymbol{w} \succcurlyeq \mathbf{0}, \mathbf{1}^\top \boldsymbol{w} = \mathbf{D}, \tag{4.1}$$

where σ^2 is the unnormalized margin variance and $\bar{\rho}$ is the unnormalized average margin. Let ρ_i denote the unnormalized margin for the ith example datum, *i.e.*,

$$\rho_i = y_i H_{i:} \boldsymbol{w}, \forall i = 1, \cdots, M. \tag{4.2}$$

In the above equations, $\boldsymbol{w}$ is the linear coefficients that weight the weak classifiers. D is the sum of these linear coefficients, which needs to be determined by cross-validation. Note that D is actually a trade-off parameter, which balances the *normalized* average margin and the *normalized* margin variance. The empirical margin variance can be computed as $\sigma^2 = \frac{1}{M-1}\sum_{i>j}(\rho_i - \rho_j)^2$. So we explicitly write the optimization in $\boldsymbol{\rho}$:

$$\min_{\boldsymbol{w}} \frac{1}{2(M-1)}\sum_{i>j}(\rho_i - \rho_j)^2 - \sum_{i=1}^{M}\rho_i, \text{ s.t. } \boldsymbol{w} \succcurlyeq 0, 1^\top \boldsymbol{w} = \mathbf{D}. \tag{4.3}$$

If we normalize the margin by setting $1^\top \boldsymbol{w} = 1$, the above problem is also equivalent to

$$\min_{\boldsymbol{w}} \frac{D}{2(M-1)}\sum_{i>j}(\rho_i - \rho_j)^2 - \sum_{i=1}^{M}\rho_i, \text{ s.t. } \boldsymbol{w} \succcurlyeq 0, 1^\top \boldsymbol{w} = 1, \tag{4.4}$$

where now $\boldsymbol{\rho}$ is the normalized margin. From this formulation, it is easy to see that D balances the two terms in the cost function. Both problems are equivalent to (4.1). We define a matrix $A \in \mathbb{R}^{M\times M}$:

$$A = \begin{bmatrix} 1 & -\frac{1}{M-1} & \cdots & -\frac{1}{M-1} \\ -\frac{1}{M-1} & 1 & \cdots & -\frac{1}{M-1} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{1}{M-1} & -\frac{1}{M-1} & \cdots & 1 \end{bmatrix}.$$

Our optimization problem can be rewritten into a simplified version:

$$\begin{aligned} &\min_{\boldsymbol{w},\boldsymbol{\rho}} \ \tfrac{1}{2}\boldsymbol{\rho}^\top A\boldsymbol{\rho} - 1^\top\boldsymbol{\rho}, \\ &\text{s.t. } \boldsymbol{w} \succcurlyeq 0, 1^\top\boldsymbol{w} = \mathbf{D}, \\ &\qquad \rho_i = y_i H_{i:}\boldsymbol{w}, \forall i = 1, \cdots, M. \end{aligned} \tag{4.5}$$

It is easy to see that A is positive semidefinite*. So (4.5) is a convex quadratic problem (QP) in $\boldsymbol{\rho}$.

If we could access all the weak classifiers (*i.e.*, the entire matrix H is given), we can solve the problem (4.5) using off-the-shelf QP solvers [4]. However, in many cases, we do not know H beforehand simply because the size of the weak classifier set could be prohibitively (or even infinitely) large. As in LPBoost [16], column-generation can be used to attack this problem. Column-generation was first proposed by [15] for solving

*A is not strictly positive definite. Since the sum of A's each row is zero, one of A's eigenvalues is zero.

some special structured linear programs with extremely large number of variables. A comprehensive survey on this technique is [49]. The general idea of column-generation is that, instead of solving the original large-scale problem (master problem), one works on a restricted master problem with a reasonably small subset of variables at each step. The dual of the restricted master problem is solved by conventional convex programming, and the optimal dual solution is used to find the new variable to be included into the restricted master problem. LPBoost is a direct application of column-generation in boosting. For the first time, LPBoost shows that, in a linear program framework, unknown weak hypotheses can be learned from the dual, although the space of all weak hypotheses is infinitely large. This is the highlight of LPBoost. This idea can be generalized to solve convex programs other than linear programming problems.[†] We next derive the dual of (4.5) such that a column-generation based optimization procedure can be devised.

4.3 The Dual of MDBoost

4.3.1 The dual formulation

The dual of a convex program always reveals some meaningful properties of the problem. We show that MDBoost is in fact a regularized version of LPBoost. The Lagrangian of (4.5) is

$$L(\underbrace{\boldsymbol{w}, \boldsymbol{\rho}}_{\text{primal}}, \underbrace{\boldsymbol{u}, r, \boldsymbol{q}}) = \tfrac{1}{2}\boldsymbol{\rho}^\top A\boldsymbol{\rho} - \mathbf{1}^\top\boldsymbol{\rho} + \mathrm{r}(\mathbf{1}^\top\boldsymbol{w} - \mathrm{D}) \\ - \boldsymbol{q}^\top\boldsymbol{w} + \textstyle\sum_{i=1}^{M} u_i(\rho_i - y_i H_{i:}\boldsymbol{w}), \tag{4.6}$$

with $\boldsymbol{q} \succcurlyeq \mathbf{0}$. The infimum of L w.r.t. to the primal variables is

$$\inf_{\boldsymbol{\rho},\boldsymbol{w}} L = \inf_{\boldsymbol{\rho}} \left[\tfrac{1}{2}\boldsymbol{\rho}^\top A\boldsymbol{\rho} + (\boldsymbol{u} - \mathbf{1})^\top\boldsymbol{\rho}\right] \\ + \inf_{\boldsymbol{w}} \left[(r\mathbf{1}^\top - \mathbf{q}^\top - \textstyle\sum_{\mathrm{i}=1}^{\mathrm{M}} \mathrm{u_i y_i H_{i:}})\boldsymbol{w}\right] - \mathrm{Dr}. \tag{4.7}$$

Clearly, $r\mathbf{1}^\top - \mathbf{q}^\top - \sum_{\mathrm{i}=1}^{\mathrm{M}} \mathrm{u_i y_i H_{i:}} = \mathbf{0}$ must hold in order to have a finite infimum. Therefore, we have

$$\textstyle\sum_{i=1}^{M} u_i y_i H_{i:} \preccurlyeq r\mathbf{1}^\top. \tag{4.8}$$

[†]Nevertheless, for linear programs, the optimal solution always lies at a vertex and column-generation solves the program exactly. For general large-scale convex programs, only an approximate solution can be found.

For the first term in L, its gradient must vanish at the optimum:

$$\frac{\partial \left[\frac{1}{2}\boldsymbol{\rho}^\top A\boldsymbol{\rho} + (\boldsymbol{u}-1)^\top \boldsymbol{\rho}\right]}{\partial \rho_i} = 0, \forall i. \tag{4.9}$$

This leads to $\boldsymbol{\rho} = -A^{-1}(\boldsymbol{u}-1)$; and the infimum is $-\frac{1}{2}(\boldsymbol{u}-1)^\top \mathbf{A}^{-1}(\boldsymbol{u}-1)$.

By putting the above results together, the dual is

$$\max_{r,\boldsymbol{u}} \; -Dr - \tfrac{1}{2}(\boldsymbol{u}-1)^\top \mathbf{A}^{-1}(\boldsymbol{u}-1), \text{ s.t. (4.8).} \tag{4.10}$$

We can reformulate (4.10) as

$$\min_{r,\boldsymbol{u}} \; r + \tfrac{1}{2D}(\mathbf{u}-1)^\top \mathbf{A}^{-1}(\mathbf{u}-1), \text{ s.t. (4.8).} \tag{4.11}$$

Under some mild conditions, weak duality and strong duality exist between the primal and dual problems we have derived. By strong duality, the two problems are equivalent. The solution of the dual gives the solution to the primal.

Note that it is critically important to keep two variables $\boldsymbol{w}$ and $\boldsymbol{\rho}$ to arrive at the dual (4.11). One may obtain a different formulation otherwise, and no column-generation based optimization can be obtained.

In our case, A is semidefinite but not strictly positive definite, and its inverse does not exist. We can replace its inverse A^{-1} with the Moore-Penrose pseudo-inverse $A^\dagger$. In our experiments, we have regularized A by $A = A + \delta\mathbf{I}$, where $\mathbf{I}$ is the identity matrix and δ is a small constant.

It is now clear that the dual problem (4.11) is a regularized hard-margin LPBoost. The second term in the cost function regularizes the dual variable $\boldsymbol{u}$. For example, when A is the identity matrix, this regularization term encourages $\boldsymbol{u}$ to approach 1. Also note that here $\boldsymbol{u}$ can take any value and it is not a distribution any more. In contrast, in AdaBoost and LPBoost, $\boldsymbol{u}$ is a distribution: $\boldsymbol{u} \succcurlyeq 0$ and $\mathbf{1}^\top \boldsymbol{u} = 1$.

The Bayesian interpretation of norm-based regularization is as follows. ℓ_2-norm assumes a Gaussian prior probability over the parameter, and ℓ_1-norm assumes a Laplacian prior probability. If we view the regularization term as the log of the probability for the parameter $\boldsymbol{x}$, we have

$$-\log p(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}^\top A^{-1}\boldsymbol{x}, & \text{if } p(\boldsymbol{x}) = \mathcal{G}(\mathbf{0}, \mathbf{A}), \\ \|\boldsymbol{x}\|_1, & \text{if } p(\boldsymbol{x}) = \prod_i \exp|x_i|, \end{cases} \tag{4.12}$$

where $\mathcal{G}(\mathbf{0}, \mathbf{A})$ is a Gaussian distribution with zero mean and covariance A. In practice, a zero-mean and unit-variance Gaussian prior is usually assumed for kernel ridge

regression, while a Laplacian prior over coefficients is typically used in sparse coding and compressed sensing.

In our case, when the number of training data is large ($M \gg 1$), A can be approximated by the identity matrix. The regularization term is simply the variance of the weights associated with each datum. Intuitively, one can design A containing useful prior information for some particular purpose.

4.3.2 Column-generation based optimization

With the above analysis, a column generalization based technique is ready to solve the problem (4.5).

Instead of solving (4.5) directly, one calculates the most violated constraint in (4.11) iteratively for the current solution and adds this constraint to the optimization problem. In theory, any column that violates dual feasibility can be added. To speed up the convergence, we add the most violated constraint by solving the following problem:

$$h'(\cdot) = \underset{h(\cdot)}{\operatorname{argmax}} \ \sum_{i=1}^{M} u_i y_i h(\boldsymbol{x}_i). \tag{4.13}$$

This is actually the same as the one that standard AdaBoost and LPBoost use for producing the best weak classifier: that is to say, to find the weak classifier that has minimum weighted training error. We summarize our MDBoost in Algorithm 4.

The convergence of Algorithm 4 is guaranteed by general column-generation or by cutting-plane algorithms, which is easy to establish. When a new $h'(\cdot)$ that violates dual feasibility is added, the new optimal value of the dual problem (maximization) would decrease. Accordingly, the optimal value of its primal problem decreases too, because they have the same optimal value due to zero duality gap. Moreover, the primal cost function is convex; therefore in the end it converges to the global minimum. Note that in the last step of the proposed MDBoost algorithm, we can get the value of $\boldsymbol{w}$ easily. Primal-dual interior-point (PD-IP) methods work on the primal and dual problems simultaneously and therefore both primal and dual variables are available after convergence. MDBoost is *totally corrective* in the sense that the coefficients of all weak classifiers are updated at each iteration.

4.4 Experiments

In this section, we run some experiments to show the effectiveness of the proposed MDBoost algorithm. In order to control the complexity of the classifier, we use the stumps as weak classifiers.

Algorithm 4 Column-generation based MDBoost.

Input: Labeled training data $(\boldsymbol{x}_i, y_i), i = 1 \cdots M$; termination threshold $\varepsilon > 0$; regularization parameter D; maximum number of iterations $N_{\max}$.

Initialization: $N = 0$; $\boldsymbol{w} = \mathbf{0}$; and $u_i = \frac{1}{M}, i = 1 \cdots M$.

for iteration $= 1 : N_{\max}$ **do**

1. Obtain a new base $h'(\cdot)$ by solving (4.13);
2. Check for optimal solution:
   if $\sum_{i=1}^{M} u_i y_i h'(\boldsymbol{x}_i) < r + \varepsilon$, **then** break;
   and the problem is solved;
3. Add $h'(\cdot)$ to the restricted master problem, which corresponds to a new constraint in the dual;
4. Solve the dual problem (4.11) and update r and u_i $(i = 1 \cdots M)$.
5. Count weak classifiers $N = N + 1$.

Output:

1. Compute the primal variable $\boldsymbol{w}$ from the optimality conditions and the last solved dual problem (primal-dual interior point methods output $\boldsymbol{w}$ as well);
2. The final strong classifier is $F(\boldsymbol{x}) = \sum_{j=1}^{N} w_j h_j(\boldsymbol{x})$.

We first show some results on a synthetic data set. 800 2D points are generated, as shown in Fig. 4.1 (top). We then run AdaBoost (1000 iterations) and MDBoost on this data set. The cumulative margin distribution is plotted in Fig. 4.1 (middle). We have set the parameter D as the sum of weak classifiers' coefficients of AdaBoost. We can see that, though MDBoost's average margin is slightly smaller than AdaBoost's average margin (0.907 *vs.* 0.938), the standard deviation of MDBoost is much smaller than that of AdaBoost (0.027 *vs.* 0.039). In this experiment, MDBoost also performs slightly better than AdaBoost (0.0375 *vs.* 0.05 with respect to test error). Note that, in terms of the minimum margin, AdaBoost is better than MDBoost. This confirms that the minimum margin is not a direct measure of test error. In Fig. 4.1 (bottom) we show the normalized value of $\boldsymbol{w}$ of the final selected weak classifiers for both algorithms. It can be seen that both algorithms select very similar decision stumps. However, the weights could be different.

Secondly, for the sake of providing a clearer insight into the feature selection of AdaBoost and MDBoost, we implement AdaBoost and MDBoost on the UCI dataset Spam whose features have explicit meanings. The iterations of training are all con-

strained below 60, which is close to the dimension of feature space. The experiment runs 20 times, and each frequency of each feature being selected by the boosting algorithms has been recorded. The average frequencies over 20 rounds are shown as a histogram in Fig. 4.2. Note that there is a cross validation (candidates for D are $\{2, 5, 8, 10, 12, 15, 20, 30, 40, 50, 70, 90, 100, 120\}$) for MDBoost. For AdaBoost, since the average test error already stopped decreasing and no overfitting is observed in the 60-th iteration, the classifier yielded in iteration 60 could be considered as optimal. As it is illustrated in Fig. 4.2, both algorithms select the obvious handy features such as "hp"(25), "free"(16) and "$"(53) with high frequencies. However, for the other features, two algorithms select them with diverse inclinations. We list the top 8 features whose frequency under AdaBoost is larger than that under MDBoost, and the top 8 features chosen by MDBoost more frequently than AdaBoost (Tab 4.1). MDBoost tends to select the features like "address", "order" and "000", which are intuitively helpful for the classification. On the contrary, the favorite ones of AdaBoost, such as "report", "email" and "conference" are more presumably irrelevant. The fact that MDBoost has smaller average test error ($11.3\% \pm 1.20$) than AdaBoost ($12.2\% \pm 1.55$) supports our analysis empirically.

		No.1	No.2	No.3	No.4	No.5	No.6	No.7	No.8
MD > Ada	Index - Name	15 - "address"	44 - "project"	29 - "lab"	40 - "direct"	51 - "["	1 - "make"	9 - "order"	23 - "000"
	Fm : Fa	0.10 : 0	0.10 : 0	0.05 : 0	0.05 : 0	0.20 : 0.05	0.40 : 0.15	0.25 : 0.10	0.45 : 0.25
MD < Ada	Index - Name	11 - "report"	31 - "telnet"	35 - "85"	18 - "email"	48 - "conference"	42 - "meeting"	36 - "technology"	17 - "business"
	Fa : Fm	0.20 : 0	0.05 : 0	0.10 : 0.05	0.55 : 0.35	0.15 : 0.10	0.40 : 0.30	0.20 : 0.15	0.65 : 0.50

Table 4.1: Most differently selected features between AdaBoost and MDBoost. "MD > Ada" means that the features are selected by MDBoost more than AdaBoost and "MD < Ada" suggests the opposite. "Index - Name" stands for the feature's index and name. "Fm" denotes the frequency of feature being selected under MDBoost while "Fa" denotes that under AdaBoost. Note that the differences are ranked by the frequency ratios (Fm : Fa or Fa : Fm).

In the third experiment, we run MDBoost on real data sets and we focus on comparing test error. We have compared four boosting algorithms: standard AdaBoost, soft-margin LPBoost [16], AdaBoost-CG[‡] and MDBoost.

The cross validation values for the parameter D for MDBoost and AdaBoost-CG are $\{2, 5, 8, 10, 12, 15, 20, 30, 40, 50, 70, 90, 100, 120\}$. The trade-off parameter C for LPBoost [16] are $\{0.001, 0.002, 0.005, 0.007, 0.01, 0.02, 0.03, 0.05, 0.07, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5\}$. The experiments are run on the 13 UCI benchmark datasets from

[‡]AdaBoost-CG is a totally corrective version of AdaBoost. It solves $\min_{\boldsymbol{w}} \log(\sum_{i=1}^{M} \exp(-y_i H_i \boldsymbol{w})), \text{s.t.}\, \boldsymbol{w} \succcurlyeq \mathbf{0}, \mathbf{1}^\top \boldsymbol{w} = \mathbf{D}$ using column-generation. See [81] for details.

[62][§]. Generally, we randomly split the data set into 3 groups. 60% of the examples are used for training; 20% are used for cross validation and the other 20% are used for test. For the data sets "ringnorm", "twonorm" and "waveform", due to their large size, only 10% examples are selected for training, cross validation and test perform on 30% and 60% examples respectively. The convergence threshold ε for LPBoost, AdaBoost-CG and MDBoost are all set to 10^{-5}. Both test and training results for the four algorithms are reported in Table 4.2 for a maximum number of iterations of 100, 500 and 1000. In some cases, the 3 totally corrective boosting algorithms (LPBoost, AdaBoost-CG, MDBoost) converge earlier than 100 iterations. We simply copy the converged results to iteration 500 and 1000, as reported in Table 4.2.

As can be seen, in terms of training error, soft-margin LPBoost demonstrates its fastest convergence in the training procedure. It finishes the column-generation iteration procedure within 100 rounds for 12 data sets but only defeats other algorithms on training error for one data set (**banana**). It might because that, compared with AdaBoost-CG, LPBoost focuses on the minimum margin rather than the margin distribution, which is harder to optimize. On the other hand, the standard AdaBoost, due to its coordinate descent optimization, convergences slowest on all the data sets but ranks the first on training error for 10 data sets. MDBoost ties with the LPBoost on training error comparison while AdaBoost-CG outperforms on 2 data sets.

In terms of test error, the proposed MDBoost outperforms on most data sets (9 among 13) and could be considered intuitively as the best algorithm with respect to generalization error. The quantitative analysis for the superiority of MDBoost will be discussed later. AdaBoost-CG has the best performance on 3 data sets. The standard AdaBoost wins the remaining one (**thyroid**). It is surprising to find that the LPBoost performs *slightly* worse than the other algorithms on all data sets. It has been observed that different LP solvers may result in slightly different performance on test data for LPBoost [97]. On some data sets, there is a significant difference between PD-IP and simplex-based solvers, in terms of iterations and the final selected weak classifiers. Here we have used Mosek [56], which implements PD-IP methods. Experiments with simplex LP solvers are needed to verify the LPBoost results. We leave this as future work. In summary, the proposed MDBoost algorithm shows competitive classification performance over AdaBoost and LPBoost. This validates the usefulness of optimizing margin distributions.

In terms of computational complexity, at each iteration MDBoost needs to solve a convex QP. The complexity of solving a QP is slightly worse than solving an LP, and

[§]`http://ida.first.fraunhofer.de/projects/bench/`

it is still very efficient. Moreover, those techniques developed for solving large-scale support vector machines may be applicable here.

In order to verify the superiority of the proposed MDBoost quantitatively, we implement three statistic tests, the Wilcoxon signed-rank test, the Friedman Test and the Bonferroni-Dunn Test, on the experimental performances of above four boosting algorithms.

The Wilcoxon signed-ranks test (WSRT)[17] is a non-parametric alternative of the paired t-test, which ranks the difference in performance of two classifiers for each data set. In this work, the WSRT is used for comparing MDBoost with the other three boosting approaches in terms of performance. The null-hypothesis declares that the concerning classifier is no better than the other boosting manners on performance. Consequently, it is a one-tail test with a conventional confidence level of 0.95. Thus, the rejection region is $\{w \in \mathbb{R} \mid w > 70\}$, considering the number of datasets with different performances is 13.[¶] Further details of WSRT are illustrated in Table 4.3. As it is shown, the null-hypothesis is rejected in the tests of MDBoost *vs*. AdaBoost, MDBoost *vs*. AdaBoost-CG and MDBoost *vs*. LPBoost. In a word, MDBoost is considered superior to other 3 boosting algorithms with respect to generalization error. AdaBoost-CG achieves slightly inferior result to MDBoost since it can not defeat MDBoost in the test. AdaBoost and LPBoost could not be considered as better than any other algorithms.

Friedman Test (FT) is a non-parametric equivalent of the repeated-measures ANOVA (Analysis of Variance) [17]. The FT can meter the difference between more than two sets of classification results. If the null-hypothesis, which assumes that all the performances are similar to each other, is rejected, a post-hoc test is processed to compared the algorithms pairwisely. The Bonferroni-Dunn Test (BDT) [17]is adopted as our post-hoc manner to verify whether a control classifier overperforms the others under the circumstance of multiple-comparison. Not surprisingly, FT rejects its null-hypothesis, which means the performances of four boosting approaches are different essentially. The confidence level for this test is also 0.95 which leads to the critical value equals to 2.291, considering the number of classifiers is 4. The result of BDT is shown in Table 4.4. According to BDT, differing from WRST, only MDBoost could be regarded as superior to both AdaBoost and LPBoost. AdaBoost-CG, on the contrary, loses the predominance to AdaBoost. The divergence might be derived from the introduction of multiple-comparison.

The result shown in Table 4.3 points to the conclusion that MDBoost is statistically

[¶]Here the critical value is not fixed since it depends on the number of datasets with different performances under two algorithms.

better than all the other competitors when compared them pairwisely and holds more obvious advantage than AdaBoost-CG when comparing with AdaBoost and LPBoost in the circumstance of multiple-comparison. From these experiments, the following three conclusions can be made.

- The convergence speed of LPBoost is the fastest algorithm among the four stepwise boosting algorithms. AdaBoost-CG and MDBoost show the similar convergence qualities and both of them are faster than AdaBoost on training procedure. This is because AdaBoost's coordinate descent nature is slow, while LPBoost, AdaBoost-CG, and MDBoost are *totally corrective*, which means at each iteration they updates all the previous weak classifier weights $\boldsymbol{w}$. This also means that with the three totally corrective boosting methods, we can use fewer weak classifiers to build a good strong classifier. This is desirable for real-time applications like face detection [91], in which the testing speed is critical.

- Our experiments confirm that a smaller training error does not necessarily lead to a smaller test error. This has been studied extensively in statistical learning theory. Take the LPBoost for example: the training quality of LPBoost is very remarkable on almost every data set while the test result can not be compared with the other two totally corrective algorithms.

- The performances of our two novel boosting algorithms, AdaBoost-CG and the MDBoost, are similar and overwhelm the other two compared algorithms. Admittedly, our approaches are slightly slower in training than the LPBoost, but they show lower generalization errors compared with LPBoost. Furthermore, as to the comparison with the standard AdaBoost, the two new algorithms illustrate better property in both training and test.

4.5 Conclusion

In the literature of boosting algorithms, there are many theoretical and empirical evidences that margin distribution plays a more important role than the minimum margin in the success of AdaBoost. However, existing boosting-like algorithms which consider margin distributions usually optimize the generalization error bound determined by margin distributions. [47, 33].

From the analysis in Chapter 3, we know that, roughly speaking, AdaBoost amounts to increases the average margin while decrease the margins' variance. Accordingly, we

can directly optimize the margin distribution: the summation over the average margin and the variance, with a trade-off parameter balancing these two terms. With some algebraic manipulations, we successfully derived the dual formulation of the optimization problem w.r.t. the margin distribution, and a column-generation-based approaches is derived for solving the dual problem. In this way, we built a analogue of LPBoost in the context of margin distribution optimization.

The proposed MDBoost inherits LPBoost's advantages such as well-defined convergence criteria, fast convergence rates and less number of weak learners in the final strong classifier. Meanwhile, MDBoost also demonstrates higher test accuracies than LPBoost, which indicates the validity of the conjecture that margin distribution is more important than the minimum margin for boosting-like algorithms. Furthermore, the derivation of MDBoost actually paves the way for solving an arbitrary quadratic problem in a boosting-like style. As an extension of this chapter, Shen *et al.* [82] used the similar strategy to optimize the loss functions of Linear Asymmetric Classification (LAC) and Fisher Linear Discriminant Analysis (FLDA). The obtained boosting-like algorithms both outperform AdaBoost in the face detection application.

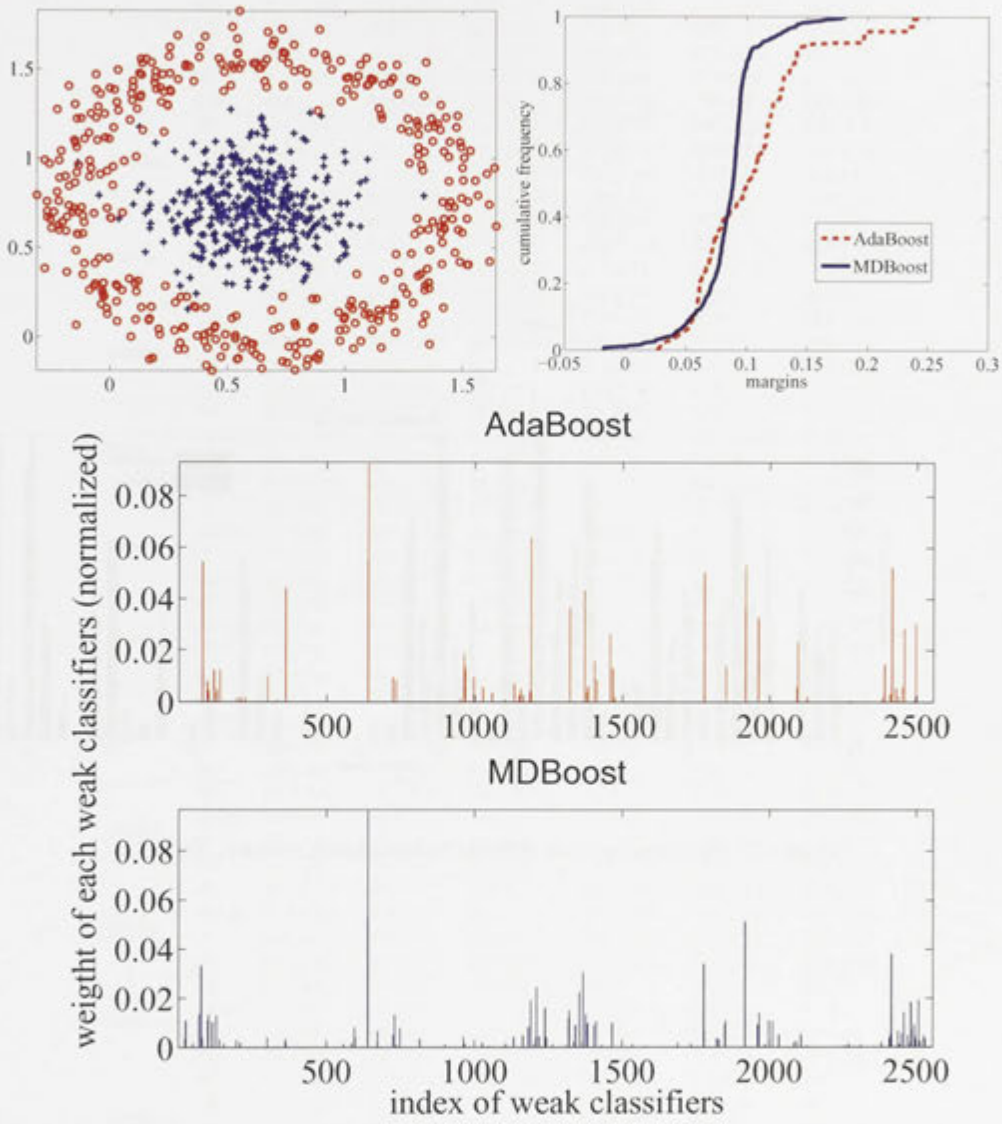


Figure 4.1: Toy data: (top-left) the data; (top-right) the cumulative frequency of margin distributions; (bottom) the normalized value of $\boldsymbol{w}$ of the final learned weak classifiers.

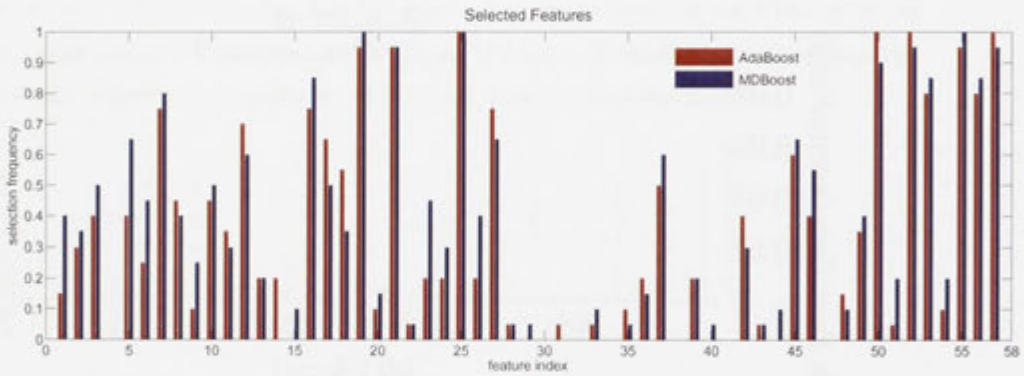


Figure 4.2: The frequencies of different features being selected, Spam.

		test error 100	test error 500	test error 1000	train error 100	train error 500	train error 1000
banana	AB	28.5 ± 1.4	28.1 ± 1.0	28.0 ± 1.1	25.9 ± 1.1	24.7 ± 0.7	24.1 ± 0.7
	AB-CG	28.0 ± 1.0	28.0 ± 1.0	28.0 ± 1.0	24.9 ± 1.0	24.9 ± 1.1	24.9 ± 1.1
	LP	37.9 ± 5.6	33.0 ± 1.9	32.2 ± 2.1	34.8 ± 5.1	$\mathbf{14.2 \pm 1.7}$	$\mathbf{7.9 \pm 7.8}$
	MD	$\mathbf{27.7 \pm 0.6}$	$\mathbf{27.7 \pm 0.7}$	$\mathbf{27.7 \pm 0.7}$	$\mathbf{22.6 \pm 2.4}$	21.9 ± 3.3	21.9 ± 3.3
b-cancer	AB	30.9 ± 5.4	31.9 ± 5.5	32.3 ± 5.6	19.9 ± 2.1	20.1 ± 1.7	20.3 ± 1.6
	AB-CG	29.4 ± 5.7	29.4 ± 5.7	29.4 ± 5.7	20.4 ± 2.0	20.4 ± 2.0	20.4 ± 2.0
	LP	34.0 ± 7.2	34.0 ± 7.2	34.0 ± 7.2	24.7 ± 4.0	24.7 ± 4.0	24.7 ± 4.0
	MD	$\mathbf{28.5 \pm 4.4}$	$\mathbf{28.5 \pm 4.4}$	$\mathbf{28.5 \pm 4.4}$	$\mathbf{19.8 \pm 2.3}$	$\mathbf{19.8 \pm 2.3}$	$\mathbf{19.8 \pm 2.3}$
diabetes	AB	24.4 ± 3.4	26.1 ± 3.8	26.8 ± 3.6	15.9 ± 1.2	$\mathbf{9.5 \pm 1.2}$	$\mathbf{5.0 \pm 1.2}$
	AB-CG	24.5 ± 3.7	24.5 ± 3.7	24.5 ± 3.7	15.5 ± 5.4	15.5 ± 5.6	15.5 ± 5.6
	LP	26.7 ± 4.4	26.4 ± 3.7	26.4 ± 3.7	$\mathbf{11.6 \pm 4.0}$	11.1 ± 4.5	11.1 ± 4.5
	MD	$\mathbf{23.8 \pm 4.0}$	$\mathbf{23.7 \pm 3.9}$	$\mathbf{23.7 \pm 3.9}$	17.0 ± 4.0	17.0 ± 4.2	17.0 ± 4.2
f-solar	AB	34.2 ± 3.5	35.2 ± 3.7	35.3 ± 3.7	33.1 ± 1.6	33.2 ± 1.7	33.2 ± 1.7
	AB-CG	34.0 ± 3.4	34.0 ± 3.4	34.0 ± 3.4	$\mathbf{32.9 \pm 1.5}$	$\mathbf{32.9 \pm 1.5}$	$\mathbf{32.9 \pm 1.5}$
	LP	34.1 ± 3.7	34.1 ± 3.7	34.1 ± 3.7	33.3 ± 1.6	33.3 ± 1.6	33.3 ± 1.6
	MD	$\mathbf{34.0 \pm 3.5}$	$\mathbf{34.0 \pm 3.5}$	$\mathbf{34.0 \pm 3.5}$	32.9 ± 1.6	32.9 ± 1.6	32.9 ± 1.6
german	AB	25.3 ± 2.9	26.6 ± 2.8	27.6 ± 2.8	18.4 ± 1.1	$\mathbf{15.7 \pm 1.2}$	$\mathbf{13.9 \pm 1.2}$
	AB-CG	$\mathbf{25.6 \pm 3.0}$	$\mathbf{25.5 \pm 3.0}$	$\mathbf{25.5 \pm 3.0}$	18.4 ± 2.6	18.2 ± 3.3	18.2 ± 3.3
	LP	29.9 ± 3.5	30.5 ± 3.5	30.5 ± 3.5	21.4 ± 8.0	15.8 ± 12.8	15.8 ± 12.8
	MD	25.7 ± 2.9	25.6 ± 2.8	25.6 ± 2.8	$\mathbf{17.5 \pm 3.0}$	17.4 ± 3.2	17.4 ± 3.2
heart	AB	19.4 ± 5.0	21.4 ± 5.8	22.0 ± 5.8	$\mathbf{3.1 \pm 1.3}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	AB-CG	17.1 ± 4.7	17.1 ± 4.7	17.1 ± 4.7	10.7 ± 3.0	10.7 ± 3.0	10.7 ± 3.0
	LP	18.5 ± 5.7	18.5 ± 5.7	18.5 ± 5.7	9.4 ± 5.9	9.4 ± 5.9	9.4 ± 5.9
	MD	$\mathbf{16.1 \pm 4.2}$	$\mathbf{16.1 \pm 4.2}$	$\mathbf{16.1 \pm 4.2}$	10.6 ± 3.1	10.6 ± 3.1	10.6 ± 3.1
image	AB	4.2 ± 0.9	3.0 ± 0.8	2.9 ± 0.9	2.4 ± 0.4	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	AB-CG	$\mathbf{3.1 \pm 0.9}$	$\mathbf{3.1 \pm 0.9}$	$\mathbf{3.1 \pm 0.9}$	$\mathbf{0.2 \pm 0.4}$	0.2 ± 0.4	0.2 ± 0.4
	LP	3.7 ± 1.2	3.2 ± 0.9	3.2 ± 0.9	0.8 ± 0.8	0.8 ± 0.8	0.8 ± 0.8
	MD	3.6 ± 0.9	3.3 ± 1.0	3.3 ± 1.0	1.8 ± 0.5	1.7 ± 0.6	1.7 ± 0.6
ringnorm	AB	7.5 ± 0.6	5.3 ± 0.4	5.2 ± 0.3	1.4 ± 0.5	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	AB-CG	7.3 ± 1.1	5.3 ± 0.3	5.3 ± 0.3	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	LP	8.3 ± 4.6	5.4 ± 0.3	5.4 ± 0.3	0.6 ± 0.4	0.3 ± 0.3	0.3 ± 0.3
	MD	$\mathbf{7.2 \pm 1.0}$	$\mathbf{5.1 \pm 0.4}$	$\mathbf{5.1 \pm 0.4}$	0.9 ± 0.3	0.3 ± 0.3	0.3 ± 0.3
splice	AB	9.2 ± 1.6	1 ± 1.7	10.4 ± 1.6	$\mathbf{5.2 \pm 1.5}$	$\mathbf{2.9 \pm 2.7}$	$\mathbf{2.7 \pm 2.8}$
	AB-CG	8.9 ± 1.3	8.9 ± 1.3	8.9 ± 1.3	6.1 ± 1.5	6.1 ± 1.5	6.1 ± 1.5
	LP	9.9 ± 2.0	10.2 ± 2.3	10.2 ± 2.3	8.1 ± 2.1	8.0 ± 2.2	8.0 ± 2.2
	MD	$\mathbf{8.2 \pm 1.0}$	$\mathbf{8.2 \pm 1.0}$	$\mathbf{8.2 \pm 1.0}$	6.2 ± 0.6	6.2 ± 0.5	6.2 ± 0.5
thyroid	AB	$\mathbf{6.7 \pm 4.3}$	$\mathbf{7.3 \pm 4.2}$	$\mathbf{7.2 \pm 4.2}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	AB-CG	7.8 ± 4.5	7.8 ± 4.5	7.8 ± 4.5	1.2 ± 2.0	1.2 ± 2.0	1.2 ± 2.0
	LP	7.8 ± 5.1	7.8 ± 5.1	7.8 ± 5.1	1.3 ± 2.0	1.3 ± 2.0	1.3 ± 2.0
	MD	7.6 ± 4.9	7.6 ± 4.9	7.6 ± 4.9	1.8 ± 1.5	1.8 ± 1.5	1.8 ± 1.5
titanic	AB	$\mathbf{21.8 \pm 1.7}$	$\mathbf{21.8 \pm 1.7}$	$\mathbf{21.8 \pm 1.7}$	$\mathbf{22.4 \pm 0.6}$	$\mathbf{22.4 \pm 0.6}$	$\mathbf{22.4 \pm 0.6}$
	AB-CG	$\mathbf{21.8 \pm 1.7}$	$\mathbf{21.8 \pm 1.7}$	$\mathbf{21.8 \pm 1.7}$	$\mathbf{22.4 \pm 0.6}$	$\mathbf{22.4 \pm 0.6}$	$\mathbf{22.4 \pm 0.6}$
	LP	21.9 ± 1.7	21.9 ± 1.7	21.9 ± 1.7	22.5 ± 1.0	22.5 ± 1.0	22.5 ± 1.0
	MD	$\mathbf{21.8 \pm 1.7}$	$\mathbf{21.8 \pm 1.7}$	$\mathbf{21.8 \pm 1.7}$	22.5 ± 0.6	22.5 ± 0.6	22.5 ± 0.6
twonorm	AB	4.2 ± 0.4	4.0 ± 0.4	4.0 ± 0.4	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	AB-CG	4.2 ± 0.4	4.1 ± 0.4	4.1 ± 0.4	0 ± 0.1	0 ± 0.1	0 ± 0.1
	LP	4.4 ± 0.3	4.3 ± 0.3	4.3 ± 0.3	0.8 ± 0.7	0.9 ± 0.7	0.9 ± 0.7
	MD	$\mathbf{3.6 \pm 0.2}$	$\mathbf{3.5 \pm 0.2}$	$\mathbf{3.5 \pm 0.2}$	0.8 ± 0.4	0.8 ± 0.3	0.8 ± 0.3
waveform	AB	12.5 ± 0.8	13.3 ± 0.8	13.6 ± 0.8	$\mathbf{0.6 \pm 0.5}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	AB-CG	$\mathbf{12.4 \pm 0.9}$	$\mathbf{12.4 \pm 0.9}$	$\mathbf{12.4 \pm 0.9}$	2.9 ± 2.0	2.9 ± 2.0	2.9 ± 2.0
	LP	12.7 ± 0.7	12.7 ± 0.6	12.7 ± 0.6	5.6 ± 2.1	5.6 ± 2.2	5.6 ± 2.2
	MD	13.0 ± 1.1	12.8 ± 0.9	12.8 ± 0.9	4.5 ± 1.7	4.5 ± 1.7	4.5 ± 1.7

Table 4.2: Test and training errors of AdaBoost (AB), AdaBoost-CG (AB-CG), LPBoost (LP) and MDBoost (MD). Most are run 50 times except for 10 times on **twonorm**, **ringnorm** and **waveform**; 20 times on **banana** due to the data sets' large sizes. The mean and standard deviation are reported. We have used decision stumps as weak classifiers. In most cases, MDBoost outperforms AdaBoost and LPBoost.

	AdaBoost	AdaBoost-CG	LPBoost	MDBoost
AdaBoost	–	No Better ($12 < 61$)	No Better ($43 < 70$)	No Better ($7 < 70$)
AdaBoost-CG	**Better** ($66 > 61$)	–	**Better** ($78 > 61$)	No Better ($18 < 70$)
LPBoost	No Better ($48 < 70$)	No Better ($0 < 61$)	–	No Better ($5 < 70$)
MDBoost	**Better** ($84 < 70$)	**Better** ($73 < 70$)	**Better** ($86 < 70$)	–

Table 4.3: Result of Wilcoxon Signed-Ranks Test (WSRT). The test is processed pairwisely among AdaBoost, AdaBoost-CG, LPBoost and MDBoost. The block where "Better" takes place indicates that the algorithm corresponding to its line is better than the algorithm corresponding to its column while "No Better" suggests the contrary. The inequality in the parenthesis is the comparison between the Wilcoxon statistic (on the left) and the critical value (on the right). The critical value depends on the number of data sets yielding different performances, thus it is not fixed. Our hypothesis will be rejected when the statistic is larger than the critical value.

	AdaBoost	AdaBoost-CG	LPBoost	MDBoost
AdaBoost	–	No Better ($-1.747 < 2.291$)	No Better ($0.75955 < 2.291$)	No Better ($-2.6584 < 2.291$)
AdaBoost-CG	No Better ($1.747 < 2.291$)	–	**Better** ($2.5065 > 2.291$)	No Better ($-0.91147 < 2.291$)
LPBoost	No Better ($-0.75955 < 2.291$)	No Better ($-2.5065 < 2.291$)	–	No Better ($-3.418 < 2.291$)
MDBoost	**Better** ($2.6584 > 2.291$)	No Better ($0.91147 < 2.291$)	**Better** ($3.418 > 2.291$)	–

Table 4.4: Result of Bonferroni-Dunn Test (BDT). Each algorithms is compared with other 3 boosting manners at the same time. The block where "Better" takes place indicates that the algorithm corresponding to its line is better than the algorithm corresponding to its column while "No Better" suggests the contrary. The inequality in the parenthesis is the comparison between the Bonferroni statistic (on the left) and the critical value (on the right). The critical value depends on the number of comparing classifiers, thus it is fixed (here is 2.291). Our hypothesis will be rejected when the statistic is larger than the critical value.

Chapter 5

Unified Framework for Regularized Risk Minimization

5.1 Introduction

In this chapter we present an unified framework for finding a linear, convex combination of weak classifiers that minimizes arbitrary loss functions with various regularization terms. Researchers have been trying to interpret the success of boosting from a few different perspectives.

Early work focused on developing theories in the framework of either probably approximately correct (PAC) learning [88] or the large margin principle [74]. Friedman *et al.* [29] developed a statistical perspective that views AdaBoost as a gradient-based stage-wise optimization method in a functional space, minimizing the exponential loss function $l(y, F) = \exp(-yF)$. AnyBoost [52, 32] generalizes this concept, in the sense that AnyBoost can optimize a broader families of loss functions. Hereafter, we refer AnyBoost to all those gradient-based boosting because their theoretical essentials are almost identical. For example, within the AnyBoost framework, one can optimize the binomial log-likelihood loss $l(y, F) = \log(1+\exp(-yF))$, which penalizes a misclassification point with less penalty than the exponential loss. Therefore, it may be more robust to outliers. In [52], a non-convex loss function has been used to have a better margin distribution, which can be translated into a smaller test error rate. Shen and Li [80] explicitly derived Lagrange dual of ℓ_1 regularized boosting for a few popular loss functions. The relationship between these dual formulations and the soft-margin LPBoost [16] was established [80].

Rosset *et al.* [68] observed that asymptotically stage-wise boosting converges to a ℓ_1 regularized solution. They deliberately set the coefficient of the weak classifier to a

very small value (ε-boost), such that the boosting method converges extremely slowly. The slowness of convergence plays the role of ℓ_1 regularization, as we will discuss in detail later. It is not new to impose regularization other than ℓ_1 in boosting. In [52], ℓ_2 norm regularized boosting has been considered and gradient-based boosting is used to perform the optimization. In [21], Duchi and Singer introduced a family of coordinate-descent methods for optimizing the upper-bounds of mixed-norm regularized boosting, based on the idea of gradient boosting [52]. Different from the conventional gradient-based boosting, they also prune the selected features that are not informative, sharing conceptual similarities with FloatBoost of Li and Zhang [45] and Zhang's forward-backward sparse learning [102]. Duchi and Singer mainly focused on learning with structural sparsity in the context of multi-class and/or multi-task applications. Both of these works are mainly inspired by gradient AnyBoost and no Lagrange duality was analyzed. Most boosting algorithms in the literature are based on the idea of AnyBoost; hence gradient-based boosting.

AnyBoost [52] is a seminal work, in the sense that it enables one to design boosting algorithms for optimizing a given cost function. It uses coordinate descent, therefore it is not totally corrective. Totally corrective boosting algorithms, like LPBoost [16], TotalBoost [97] and those proposed in [80], update the coefficients of all the past selected weak learners at each iteration. The totally corrective boosting algorithms usually need a lower number of iterations to achieve convergence [80]. Beside the convergence speed, the AnyBoost framework did not realize that many boosting algorithms such as AdaBoost are actually ℓ_1 norm regularized. Slow convergence, which is the nature of coordinate descent, plays the role of selecting the ℓ_1 norm regularization parameter [68]. Without seeing this, it could be difficult to explain why AnyBoost works. For example, when all the training examples are separable, AdaBoost's objective function $\sum_i \exp(-\sum_j y_i w_j h_j(\boldsymbol{x}_i))$ is not well defined. Indeed, one can always make the objective arbitrarily approach zero by multiplying a positive factor to $\boldsymbol{w}$. In contrast, we explicitly put boosting learning into the regularized empirical risk minimization framework and use convex optimization tools to analyze its characteristics. We will see that the only difference between boosting and kernel learning is the optimization procedure. If all the weak learners were given, there would be no essential difference between boosting and kernel methods. The most important component of our work is to propose a general and totally corrective boosting learning framework that can be used to minimize regularized risk with arbitrary convex loss functions and arbitrary convex regularization terms in the form of problem (5.1).

The main contributions of this chapter follow.

1. We generalize the totally corrective ℓ_1 regularized boosting algorithms in [80] to *arbitrary* convex loss functions. We also show that a few variants of boosting algorithms in the literature can be interpreted in the proposed framework.

2. We propose a very general framework that can accommodate *arbitrary* convex regularization terms other than ℓ_1 norm. By explicitly deriving the Lagrange dual formulations, we demonstrate that totally corrective boosting based on column-generation can be designed to facilitate boosting. In particular, we focus on analyzing the ℓ_1, ℓ_2, and ℓ_∞ norm regularization.

3. By introducing the concept of the nonnegatively clipped edge of a weak classifier, we show the connection of ℓ_1, ℓ_2, and ℓ_∞ norm regularized boosting. The Lagrange dual formulations of these can be interpreted in a unified framework.

4. We also show how the Fenchel dual of a convex loss function in the primal regularizes the dual variable (the training samples' importance weights). Hence we generalize the results in [80], where only the exponential loss, logistic loss and generalized hinge loss are considered. We now know that the Fenchel dual of an arbitrary convex loss penalizes the divergence of the sample weights.

5. For the first time, we observe that the totally corrective boosting's primal problems are much simpler than the counterpart dual problems. So at each iteration of a column-generation based boosting algorithm, it is much faster to solve the primal problem. In the proposed AnyBoost$_{\mathrm{TC}}$, we do not generally require sophisticated convex solvers and only gradient descent methods like L-BFGS-B [104] are needed. Previous totally corrective boosting algorithms [16, 80, 97] all solve the dual problems using convex optimization solvers. Besides the primal problems' much simpler structures, the dual problems, in most cases, have many more variables than their corresponding primal problems.

The remainder of the chapter is organized as follows. Before presenting the main results, we introduce the basic idea of boosting and the Fenchel conjugate in Section 5.2. In Section 5.3, we extend the results in [80] to arbitrary convex loss functions. A new totally corrective boosting is proposed to minimize the ℓ_1 regularized risk. The connection of the proposed algorithm to some previous boosting algorithms is discussed. In Section 5.4, we generalize the totally corrective boosting to arbitrary convex regularization. We also briefly discuss boosting for regression. We present experimental results in Section 5.5 and conclude the chapter in the last section.

5.2 Preliminaries

In statistics or signal processing, when we face ill-posed problems, regularization is needed to enforce stability of the solution. Regularization usually improves the conditioning of the problem. Literature on this subject is immense. In statistical learning, in particular, supervised learning, one learns a function that best describes the relation between input $\boldsymbol{x}$ and output y. Statistical learning theory tells us that often a regularization term is needed for a learning machine in order to trade off the training error and the generalization capability [89].

Concretely, we solve the following problem for training a classifier or a regressor:

$$\inf_{F \in \mathcal{F}} \sum_{i=1}^{m} l(F(\boldsymbol{x}_i), y_i) + \nu \Omega(F(\cdot)). \tag{5.1}$$

Here $l(\cdot)$ is a data-fitting loss function, which corresponds to the empirical risk measure, and $\Omega(\cdot)$ is a regularization function. So the first term corresponds to the empirical risk. The parameter $\nu \geq 0$ balances these two terms. $\mathcal{F}$ is the functional space in which the classification function $F(\cdot)$ resides. Clearly, without the regularization term, if $\mathcal{F}$ is very large, it can easily lead to over-fitting and the minimizers can be nonsense. The regularized formulation considers the trade-off between the quality of the approximation over the training data and the complexity of the approximating function [89]. Often simplicity is manifested as sparsity in the solution vector—or some transformation of it. Typically, ℓ_p norm functions can be used for regularization, such as ℓ_1 norm in Lasso [85], ℓ_2 norm in ridge regression, and RKHS regularization in kernel methods*. ℓ_1 norm regularization may create sparse answers and better approximations in relevant cases. ℓ_1 norm regularization methods have recently gained much attention in compressed sensing [10] and machine learning, due to the induced sparsity and to being easy to optimize as a surrogate of the non-convex ℓ_0 pseudo-norm [58, 100].

For boosting algorithms, $F(\cdot)$ takes the form

$$F(\boldsymbol{x}) = \sum_{i=1}^{n} w_i h_i(\boldsymbol{x}), \tag{5.2}$$

with $\boldsymbol{w} \succcurlyeq 0$. This nonnegativeness constraint can always be enforced because one can flip the sign of the weak classifier $h(\cdot)$.

It has been shown that some boosting algorithms can be viewed as ℓ_1 norm regu-

*Due to the representer theorem, RKHS regularization is special: the optimal solution in a high-dimensional functional space can be found by solving a finite dimensional minimization problem.

larized model fitting [68]. We can rewrite the learning problem into

$$\inf_{F\in\mathcal{F}} \sum_{i=1}^{m} l(\gamma_i) + \nu \mathbf{1}^\top \boldsymbol{w} \tag{5.3}$$

where γ_i is the unnormalized margin: $\gamma_i = y_i F(\boldsymbol{x}_i) = y_i H_{i:} \boldsymbol{w}$.

As an important concept in this thesis, let us recall the definition of Fenchel duality again.

Definition 5.2.1. *(Fenchel duality) Let $f : \mathbb{R}^n \to \mathbb{R}$. The function $f^* : \mathbb{R}^n \to \mathbb{R}$, defined as*

$$f^*(\boldsymbol{u}) = \sup_{\boldsymbol{x}\in \mathrm{dom}\, f} (\boldsymbol{u}^\top \boldsymbol{x} - f(\boldsymbol{x})), \tag{5.4}$$

is called the Fenchel duality of the function $f(\cdot)$. The domain of the conjugate function consists of $\boldsymbol{u} \in \mathbb{R}^n$ for which the supremum is finite.

$f^*(\cdot)$ is always a convex function because it is the point-wise supremum of a family of affine functions of $\boldsymbol{u}$, even if $f(\cdot)$ is non-convex [4]. If $f(\cdot)$ is convex and closed, then $f^{**} = f$. For a point-wise loss function, $l(\boldsymbol{\gamma}) = \sum_{i=1}^m l(\gamma_i)$, the Fenchel duality of the sum is the sum of the Fenchel dualities:

$$\begin{aligned} l^*(\boldsymbol{u}) &= \sup_{\boldsymbol{\gamma}} \left\{ \boldsymbol{u}^\top \boldsymbol{\gamma} - \sum_{i=1}^{m} l(\gamma_i) \right\} = \sum_{i=1}^{m} \sup_{\gamma_i} \{u_i \gamma_i - l(\gamma_i)\} \\ &= \sum_{i=1}^{m} l^*(u_i). \end{aligned}$$

Clearly, the shape of $f^*(u)$ is determined by $f(x)$ and *vice versa*. We consider functions of Legendre type [67] in this work. That means the gradient $f'(\cdot)$ is defined on the domain of $f(\cdot)$ and is an isomorphism between the domains of $f(\cdot)$ and $f^*(\cdot)$. If $f(\cdot)$ admits a strict supporting line at x with slope u, then $f^*(\cdot)$ admits a tangent supporting line at u with slope $f^{*\prime}(u) = x$.

5.3 ℓ_1 Norm Regularized AnyBoost$_{\mathrm{TC}}$

The general ℓ_1 regularized optimization problem we want to solve is

$$\begin{aligned} &\min_{\boldsymbol{w},\boldsymbol{\gamma}} \ \sum_{i=1}^{m} l(\gamma_i) + \nu \cdot \mathbf{1}^\top \boldsymbol{w} \\ &\text{s.t. } \gamma_i = y_i H_{i:} \boldsymbol{w} \ (\forall i = 1 \cdots m), \ \boldsymbol{w} \succcurlyeq 0. \end{aligned} \tag{5.5}$$

Name	Loss $l(F, y)$	Derivative $l'(F, y)$
Exponential	$\exp(-yF)$	$-y\exp(-yF)$
Logistic	$\log(1+\exp(-yF))$	$-y/(1+\exp(yF))$
Hinge	$\max(0, -yF)$	0 if $yF \geq 0$; $-y$ otherwise
Squared hinge	$0.5[\max(0, -yF)]^2$	0 if $yF \geq 0$; F otherwise
MadaBoost loss [20]	$\exp(-yF)$ if $yF \geq 0$, otherwise $1 - yF$	$-y\exp(-yF)$ if $yF \geq 0$; $-y$ otherwise
Least square	$0.5(y-F)^2$	$F - y$
ℓ_1 norm	$\lvert y - F\rvert$	$\operatorname{sgn}(F-y)$
Huber's loss	$0.5(y-F)^2$ if $\lvert y-F\rvert < 1$, otherwise $\lvert y-F\rvert - 0.5$	$F - y$ if $\lvert y-F\rvert < 1$; $\operatorname{sgn}(F-y)$ otherwise
Poisson regression	$\exp(F) - yF$	$\exp(F) - y$
Quantile regression	$\max(\tau(F-y), (1-\tau)(y-F))$	τ if $F > y$; $\tau - 1$ otherwise
ε-insensitive regression	$\max(0, \lvert y-F\rvert - \varepsilon)$	0 if $\lvert y-F\rvert \leq \varepsilon$; $\operatorname{sgn}(F-y)$ otherwise

Table 5.1: Loss functions and their derivatives.

We now derive its Lagrange dual problem. Although the variable of interest is $\boldsymbol{w}$, we keep the auxiliary variable $\boldsymbol{\gamma}$ in order to derive a meaningful dual. The Lagrangian is

$$\begin{aligned} L &= \sum_{i=1}^{m} l(\gamma_i) + \nu \mathbf{1}^\top \boldsymbol{w} - \boldsymbol{u}^\top (\boldsymbol{\gamma} - \operatorname{diag}(\boldsymbol{y}) H \boldsymbol{w}) - \boldsymbol{p}^\top \boldsymbol{w} \\ &= \left(\nu \mathbf{1}^\top + \boldsymbol{u}^\top \operatorname{diag}(\boldsymbol{y}) H - \boldsymbol{p}^\top\right) \boldsymbol{w} - \left(\boldsymbol{u}^\top \boldsymbol{\gamma} - \sum_{i=1}^{m} l(\gamma_i)\right), \end{aligned}$$

with $\boldsymbol{p} \succcurlyeq 0$. To find its infimum over the primal variables $\boldsymbol{w}$ and γ, we must have

$$\nu \mathbf{1}^\top + \boldsymbol{u}^\top \operatorname{diag}(\boldsymbol{y}) H - \boldsymbol{p}^\top = \mathbf{0},$$

which leads to

$$\boldsymbol{u}^\top \operatorname{diag}(\boldsymbol{y}) H \succcurlyeq -v \mathbf{1}^\top; \tag{5.6}$$

and

$$\inf_{\boldsymbol{w}, \gamma} L = -\sup_{\gamma} \boldsymbol{u}^\top \boldsymbol{\gamma} - \sum_{i=1}^{m} l(\gamma_i) = -\sum_{i=1}^{n} l^*(u_i).$$

Therefore, the dual problem is

$$\min_{\boldsymbol{u}} \ \sum_{i=1}^{m} l^*(u_i), \quad \text{s.t. (5.6).} \tag{5.7}$$

We can reverse the sign of $\boldsymbol{u}$ and rewrite (5.7) into its equivalent form

$$\min_{\boldsymbol{u}} \ \sum_{i=1}^{m} l^*(-u_i), \tag{5.8a}$$

$$\text{s.t.} \ \ \boldsymbol{u}^\top \operatorname{diag}(\boldsymbol{y}) H \preccurlyeq \nu \mathbf{1}^\top. \tag{5.8b}$$

From the Karush-Kuhn-Tucker (KKT) conditions, between the primal (5.5) and the dual (5.8) the relationship

$$u_i = -l'(\gamma_i), \forall i, \tag{5.9}$$

holds at optimality. This means that the weight u_i associated to each sample is the negative gradient of the loss at γ_i. This can be easily obtained by setting the first derivative of L w.r.t. γ_i to zeros. Under the assumption that both the primal and dual problems are feasible and the Slater's condition satisfies, strong duality holds between (5.5) and (5.8), which means that their solutions coincide such that one can obtain both solutions by solving either of them.

If we know all the weak classifiers, that is, the matrix H can be computed a priori, the original problem (5.5) can be easily solved (at least in theory) because it is an optimization problem with simple nonnegativeness constraints. In practice, however, we usually cannot compute all the weak classifiers since the size of the weak classifier set $\mathcal{H}$ could be prohibitively large or even infinite. In convex optimization, column-generation (CG) is a technique that can be used to attack this difficulty. The crucial insight behind CG is that for a linear program, the number of non-zero variables of the optimal solution is equal to the number of constraints, hence although the number of possible variables may be large, we need only a small subset of these in the optimal solution. For a general convex problem, CG can still be used to obtain an approximate solution. It works by considering only a small subset of the entire variable set. Once it is solved, we ask the question: "Are there any other variables that can be included to improve the solution?" So we must be able to solve the subproblem: given a set of dual values, one either identifies a variable that has a favorable reduced cost, or indicates that such a variable does not exist. In essence, CG finds the variables with negative reduced costs without explicitly enumerating all variables.

We now consider only a small subset of the variables in the primal: *i.e.*, only a subset of $\boldsymbol{w}$ is used. The problem solved using this subset is usually termed the restricted master problem (RMP). Because the primal variables correspond to the dual constraints, solving RMP is equivalent to solving a relaxed version of the dual problem. With a finite $\boldsymbol{w}$, the set of constraints in the dual (5.8) is finite, as each constraint corresponds to a primal variable w_i, $\forall i$. And we can solve (5.8) that satisfies all the existing constraints. If we can prove that among all the constraints that we have not added to the dual problem, no single constraint is violated, then we can conclude that solving the restricted problem is equivalent to solving the original problem. Otherwise, there exists at least one constraint that is violated. The violated constraints correspond to variables in primal that are not in RMP. Adding these variables to RMP leads to a

new RMP that needs to be re-optimized.

The general algorithm to solve our boosting optimization problem using CG—hence the name AnyBoost$_{\mathrm{TC}}$—is summarized in Algorithm 5. A few comments on AnyBoost$_{\mathrm{TC}}$ are:

1. Practically, we set the stopping criterion as $\sum_{i=1}^{m} u_i y_i h'(\boldsymbol{x}_i) \leq \nu + \varepsilon$ where ε is a small user-specified constant;

2. The core part of AnyBoost$_{\mathrm{TC}}$ is the update of $\boldsymbol{u}$ (Line 5). Standard CG in convex optimization typically solves the dual problem. In our case, the dual problem (5.8) is a convex program that has m variables and n constraints. The primal problem (5.5) has n variables and n simple constraints (the equality constraints are only for deriving the dual and can be put back to the cost function). In boosting, often we have more training examples than final weak classifiers. That is, $m \geq n$. Moreover, n increases by one at each iteration. At the beginning, only small-scale problems are involved in the primal (5.5). As we will show, the simple constraints in (5.5) are also much easier to cope with. Quasic-Newton algorithms like L-BFGS-B [104] can be used to solve (5.5). In contrast, usually sophisticated primal-dual interior-point based algorithms are needed for solving the convex problem (5.8). All in all, (5.5) is easier to solve. Given $\boldsymbol{w}$, we can calculate $\boldsymbol{u}$ via the optimality condition (5.9). We do not need to know the Fenchel dual of the loss $l^*(\cdot)$ explicitly. We list some popular classification and regression loss functions and their first derivatives in Table 5.2.

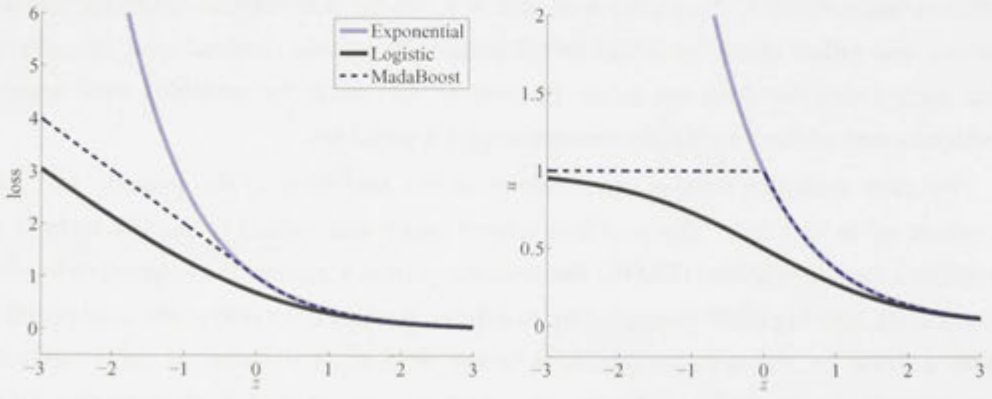


Figure 5.1: Some loss functions (first) and their first derivatives (second). Here the cost function of MadaBoost is defined as $l(z) = \exp(-z)$ if $z > 0$; $1 - z$ otherwise.

Algorithm 5 terminates after a finite number of iterations at a global optimum. Theorem 5.1 guarantees the convergence of Algorithm 5. Generally, the CG method's

Algorithm 5 ℓ_1 norm regularized AnyBoost$_{\text{TC}}$ for classification.

Input: Training data $\{(\boldsymbol{x}_i, y_i)\}, i = 1 \cdots m$; a convergence threshold $\varepsilon > 0$.

Initialization: $\boldsymbol{w} = 0, \boldsymbol{u} = \frac{1}{m}\mathbf{1}$.

while true **do**

- Receive a weak classifier that most violates the dual constraint:

$$\hat{h}(\cdot) = \operatorname*{argmax}_{h(\cdot)} \sum_{i=1}^{m} u_i y_i h(\boldsymbol{x}_i);$$

- Check for the stopping criterion:

if $\sum_{i=1}^{m} u_i y_i \hat{h}(\boldsymbol{x}_i) \leq \nu + \varepsilon$, **then** break;

- Add $\hat{h}(\cdot)$ into the primal problem that corresponds to a new variable;[a]
- Obtain $\boldsymbol{w}$ by solving the primal (5.5) and also using (5.9) to update the dual variable $\boldsymbol{u}$.

Output: Output a convex combination of the weak classifiers.

[a] We can also add $\hat{h}(\cdot)$ to the dual problem as a new constraint. Then one solves the dual problem in the next step.

convergence follows by standard CG algorithms in convex optimization. We include this theorem for self-completeness.

Theorem 5.1. *Assume that we can exactly solve the subproblem* $\hat{h}(\cdot) = \operatorname*{argmax}_{h(\cdot)} \sum_{i=1}^{m} u_i y_i h(\boldsymbol{x}_i)$ *at each iteration, then Algorithm 5 either halts on the round that the stopping criterion is met* $\sum_{i=1}^{m} u_i y_i \hat{h}(\boldsymbol{x}_i) \leq \nu + \varepsilon$, *up to the desired accuracy* ε, *or converges to some finite value.*

The convergence follows the general column-generation based technique in convex optimization, although the convergence rate is not known.

From the KKT condition (5.9) we can derive some interesting results.

Result 5.3.1. *At each iteration of LPBoost [16], a sample* $\boldsymbol{x}_i$ *with a negative margin* γ_i *(*i.e.*, miss-classified) will have a nonzero weight* u_i*; those samples that have positive margins (correctly classified) will all have zero weights, and they are not considered in the next iteration.*

We now show that AdaBoost and other boosting algorithms are just specific cases of AnyBoost$_{\text{TC}}$ in Algorithm 5, for the appropriate choice of the regularization parameter ν, loss function and optimization strategy.

Theorem 5.2. *AdaBoost $_\rho$ [64] minimizes the regularized AdaBoost's cost function with the regularization parameter $\nu = \rho$ via coordinate-descent.*

AdaBoost is a special case of AnyBoost$_{\text{TC}}$ with a very small regularization parameter ν (ν approaching zero), and a coordinate-descent optimization strategy to minimize the primal problem (Step (3) of AnyBoost$_{\text{TC}}$).

Proof: To prove that AdaBoost is indeed AnyBoost$_{\text{TC}}$ with loss $l(y, F) = \exp(-yF)$, let us examine each step of AnyBoost$_{\text{TC}}$. Clearly, Step (1) of AnyBoost$_{\text{TC}}$ is identical with AdaBoost. For the stopping criterion, when $\nu \to 0$, it is easy to verify that both algorithms stop at iteration $t+1$ when

$$\varpi_+ - \varpi_- < 0;$$

with

$$\varpi_+ = \sum_{i:y_i h_{t+1}(\boldsymbol{x}_i)>0} u_i^t,$$

$$\varpi_- = \sum_{i:y_i h_{t+1}(\boldsymbol{x}_i)<0} u_i^t;$$

i.e.[†], the weighted error is larger than the weighted accuracy. Here we assume a discrete AdaBoost. It is straightforward to extend to real-valued $h(\cdot) \in [-1, +1]$ as discussed in [76].

If we adopt coordinate-descent to optimize the primal at Step (3), at iteration $t+1$, we keep $w_1, \cdots, w_t$ fixed. Given the chosen $h_{t+1}(\cdot)$ we want to find w_{t+1} that minimizes

$$\begin{aligned} C_{\exp} &= \sum_{i=1}^{m} \exp(-\gamma_i^t) \cdot \exp(-y_i w_{t+1} h_{t+1}(\boldsymbol{x}_i)) + \nu \mathbf{1}^\top \boldsymbol{w} \\ &= \sum_{i=1}^{m} u_i^t \exp(-y_i w_{t+1} h_{t+1}(\boldsymbol{x}_i)) + \nu w_{t+1}, \\ &= \varpi_+ \exp(-w_{t+1}) + \varpi_- \exp(w_{t+1}) + \nu w_{t+1}, \end{aligned} \tag{5.10}$$

subject to $w_{t+1} > 0$. we have dropped the terms that are determined by the fixed variables $w_1, \cdots, w_t$. Here we have used the fact from (5.9) that

$$u_i = \exp(-\gamma_i), \forall i, \tag{5.11}$$

to minimize $c_{\exp}$, set its first derivative to zero:

$$-\varpi_+ \exp(-w_{t+1}) + \varpi_- \exp(w_{t+1}) + \nu = 0. \tag{5.12}$$

[†]Hereafter, subscript t indexes the iteration of AnyBoost$_{\text{TC}}$.

a close-form solution for w_{t+1} is:

$$w_{t+1} = \log\left(\sqrt{\frac{\varpi_+}{\varpi_-} + \frac{\nu^2}{4\varpi_-^2}} - \frac{\nu}{2\varpi_-}\right). \tag{5.13}$$

when ν is negligible, we have a solution for w_{t+1}:

$$w_{t+1} = \frac{1}{2}\log\frac{\varpi_+}{\varpi_-}, \tag{5.14}$$

which is consistent with AdaBoost. The rule for updating $\boldsymbol{u}$ can be trivially seen from (5.11).

Note that in the above analysis, we do not need to normalize $\boldsymbol{u}$. This is different from AdaBoost. Actually if we replace the entire loss $\sum_{i=1}^m \exp(-\gamma_i)$ with its logarithmic version $\log(\sum_{i=1}^m \exp(-\gamma_i))$; *i.e.*, we minimize $\log(\sum_{i=1}^m \exp(-\gamma_i)) + \nu \mathbf{1}^\top \boldsymbol{w}$, we have

$$u_i = \frac{\exp(-\gamma_i)}{\sum_{i=1}^m \exp(-\gamma_i)}, \forall i,$$

which results in $\mathbf{1}^\top \boldsymbol{u} = 1$. the cost (5.10) becomes $c_{\exp} = \log(\varpi_+ \exp(-w_{t+1}) + \varpi_- \exp(w_{t+1})) + \nu w_{t+1}$, where we have dropped $\log(\sum_{i=1}^m \exp(-\gamma_i^t))$ that is independent of w_{t+1}. it is easy to see that

$$w_{t+1} = \frac{1}{2}\left(\log\frac{\varpi_+}{\varpi_-} - \log\frac{1+\nu}{1-\nu}\right) \tag{5.15}$$

minimizes the new log-sum-exp cost function. This is the rule used in AdaBoost $_\rho$ [64]. Clearly when ν is small, (5.14) and (5.15) coincide. So, by simply replacing the update rule in AdaBoost with (5.15), we get an explicitly ℓ_1-norm regularized AdaBoost. □

We now know that AdaBoost is indeed a ℓ_1-norm regularized algorithm [68]. It is mysterious that AdaBoost does not have any parameter to tune and it works so well on many datasets. We have shown that AdaBoost simply sets the regularization parameter ν to a very small value. Note that one cannot set the regularization parameter ν to zero. Without this regularization term, the problem (5.1) is ill-posed. For the AdaBoost and logistic boosting losses, on separable data, one can always make the first term of (5.1) approach zero by multiplying an arbitrarily large positive factor to $\boldsymbol{w}$. In this scenario, it is unlikely to obtain a reasonable $\boldsymbol{w}$ by minimizing either of the losses.

We conjecture that a carefully-selected ν would yield better performances, especially on noisy datasets. This may partially explain why AdaBoost over-fits on noisy datasets. However, early stopping for AdaBoost eliminates over-fitting to some extend [103].

For the normalized version (log-sum-exp loss), we have a simpler closed-form update rule and no computation overhead is introduced compared with the standard AdaBoost. From (5.15), ν must be less than 1; hence $0 < \nu < 1$. As long as the selected weak classifier $h_{t+1}(\cdot)$ does not satisfy the stopping criterion, *i.e.*, $\varpi_+ - \varpi_- > \nu$, w_{t+1} calculated with (5.15) must be positive. Clearly a larger ν makes AnyBoost$_{\text{TC}}$ converge faster.

Strategies such as *shrinkage* [29] and *bounded step-size* [103] have been proposed to preventing over-fitting. It is well known that these methods are other forms of regularization. In AdaBoost, shrinkage corresponds to replacing the w_{t+1} with $\eta' w_{t+1}$ where $0 < \eta' < 1$; while bounded step-size caps w_{t+1} by $\min\{w_{t+1}, \eta''\}$ where η'' is a small value. Both of these two methods decrease the step-size for producing better generalization performances. Starting from the general regularized statistical learning machine (5.1), we are able to show that the regularized AdaBoost takes the form of (5.15) for updating the step-size. The fundamental idea is consistent with the two previous heuristics. Arc-Gv [8], proposed for producing larger minimum margins, modifies AdaBoost's updating rule as

$$w_{t+1} = \frac{1}{2}\left(\log\frac{\varpi_+}{\varpi_-} - \log\frac{1+\gamma'_t}{1-\gamma'_t}\right), \tag{5.16}$$

where γ'_t is the normalized minimum margin over all training samples of the combined classifier up to iteration t: $\gamma'_t = \min_i\{y_i \sum_{j=1}^t w_j h_j(\boldsymbol{x}_i)/\sum_{j=1}^t w_j\}$. Comparing (5.15) and (5.16), we have the following corollary.

Corollary 5.3.1. *Arc-Gv [8] is a regularized version of AdaBoost with an adaptive regularization parameter, which is the normalized minimum margin over all training examples.*

From the viewpoint of regularization theory, there is no particular reason that we should relate the regularization parameter ν to the minimum margin. Arc-Gv's purpose is to maximize the minimum margin to the extreme, which has been shown not beneficial for the final performance [65].

We can also design a totally correctively AdaBoost easily according to the AnyBoost$_{\text{TC}}$ framework. As described in Algorithm 5, we can optimize either the dual or the primal. With the log-sum-exp loss, the dual problem of AdaBoost is

$$\min_{\boldsymbol{u}} \sum_{i=1}^m u_i \log u_i, \;\; \text{s.t. (5.8b) and } \mathbf{1}^\top \boldsymbol{u} = 1, \boldsymbol{u} \succcurlyeq \mathbf{0}, \tag{5.17}$$

which is an entropy maximization problem. This is a general constrained convex program. It can be solved using primal-dual interior point algorithms like [56]. Alternatively we can also solve it in the primal. We use L-BFGS-B [104] to solve the primal

problem. L-BFGS-B is faster and more scalable. TotalBoost [97] takes the same form as (5.17), except that the parameter ν is adaptively set to the minimum edge over all weak classifiers generated up to the current iteration. It is clear now that TotalBoost is also a regularized AdaBoost:

Corollary 5.3.2. *AdaBoost$^*_\nu$ [64] and TotalBoost [97] minimize the regularized versions of AdaBoost's loss with an adaptive regularization parameter, which is the minimum edge over all weak classifiers (up to a numerical accuracy ν).*

AdaBoost$^*_\nu$ employs an coordinate descent optimization strategy while TotalBoost optimizes the cost function totally correctively.

Again it remains unclear whether ν should be related to the minimum edge and how it is translated into the final generalization performance, although it is very clear that the minimum margin is efficiently maximized in AdaBoost$^*_\nu$ and TotalBoost.

TotalBoost [97] failed to realize the primal-dual relationship of AdaBoost's cost function and (5.17), and an LPBoost is solved to obtain the final strong classifier after iteratively solving (5.17). In other words, in TotalBoost, the dual problem (5.17) is only used to generate weak classifiers.

As a conclusion of this section, we highlight that ℓ_1 norm regularized AnyBoost$_{\mathrm{TC}}$ in Algorithm 5 is consistent with AnyBoost of [52].

Theorem 5.3. *In Algorithm 5, if we set the regularization parameter $\nu = 0$, and solve the primal problem using coordinate descent (Line 5 of Algorithm 5),* i.e., *keep $w_1, \ldots, w_j$ fixed at iteration $j+1$, then Algorithm 5 is the same as AnyBoost of Mason* et al. *[52]. Therefore, AnyBoost can be seen as a special form of Algorithm 5.*

Proof: The proof shares similarities with the proof of Theorem 5.2. It is straightforward to establish this connection. □

Mason *et al.*'s AnyBoost is not immediately applicable to regression while our AnyBoost$_{\mathrm{TC}}$ can be used for regression without any modification.

5.4 A More General Formulation

Let us consider a more general formulation of the optimization problem (5.5):

$$\begin{aligned} &\min_{\boldsymbol{w},\boldsymbol{\gamma}} \ \sum_{i=1}^{m} l(\gamma_i) \\ &\text{s.t.}\, Q\boldsymbol{w} \leq \boldsymbol{r}, \gamma_i = y_i H_{i:}\boldsymbol{w}\,(\forall i = 1 \ldots m), \boldsymbol{w} \succcurlyeq 0, \end{aligned} \tag{5.18}$$

where $Q \in \mathbb{R}^{p\times n}$ and $\boldsymbol{r} \in \mathbb{R}^p$ encode the regularization term and prior information when available. it is trivial to show (5.18) covers (5.5) as a special case. If we take $Q = \mathbf{1}^\top \in \mathbb{R}^{1\times n}$ and write the first constraint as $\mathbf{1}^\top \boldsymbol{w} \leq r$. We know that for a certain ν, one can always find a r such that the solution of (5.5) solves (5.18) too.

The Lagrange dual of (5.18) is

$$\min_{\boldsymbol{u},\boldsymbol{s}} \sum_{i=1}^{m} l^*(-u_i) + \boldsymbol{r}^\top \boldsymbol{s} \tag{5.19a}$$

$$\text{s.t. } \boldsymbol{u}^\top \operatorname{diag}(\boldsymbol{y}) H \preccurlyeq \boldsymbol{s}^\top Q, \tag{5.19b}$$

$$\boldsymbol{s} \succcurlyeq \mathbf{0}. \tag{5.19c}$$

Here the dual variables are $\boldsymbol{u} \in \mathbb{R}^m$ and $\boldsymbol{s} \in \mathbb{R}^p$.

The optimality condition (5.9) holds too.

The ℓ_∞ norm regularization is also a special case of the above formulation. ℓ_∞-norm regularization has been used in kernel classifiers [106]. If we let $Q = \mathbf{I} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{r} = r\mathbf{1}$, this is $\|\boldsymbol{w}\|_\infty \leq r$.

Let us have a close look at the ℓ_∞-norm regularized boosting. The ℓ_∞-norm regularized boosting can be written as

$$\min_{\boldsymbol{w},\boldsymbol{\gamma}} \sum_{i=1}^{m} l(\gamma_i) \tag{5.20a}$$

$$\text{s.t. } \mathbf{0} \preccurlyeq \boldsymbol{w} \preccurlyeq r\mathbf{1},\ \gamma_i = y_i H_{i:}\boldsymbol{w}\ (\forall i = 1 \dots m). \tag{5.20b}$$

The Lagrangian is

$$\begin{aligned} L &= \sum_{i=1}^{m} l(\gamma_i) + \boldsymbol{s}^\top(\boldsymbol{w} - r\mathbf{1}) - \boldsymbol{q}^\top \boldsymbol{w} - \boldsymbol{u}^\top(\boldsymbol{\gamma} - \operatorname{diag}(\boldsymbol{y}) H \boldsymbol{w}) \\ &= \left(\boldsymbol{s}^\top + \boldsymbol{u}^\top \operatorname{diag}(\boldsymbol{y}) H - \boldsymbol{q}^\top\right) \boldsymbol{w} - \left(\boldsymbol{u}^\top \boldsymbol{\gamma} - \sum_{i=1}^{m} l(\gamma_i)\right) \\ &\quad - r\mathbf{1}^\top \boldsymbol{s}, \end{aligned}$$

with $\boldsymbol{q} \succcurlyeq 0$ and $\boldsymbol{s} \succcurlyeq 0$.

Therefore, its corresponding Lagrange dual is

$$\min_{\boldsymbol{u},\boldsymbol{s}} \sum_{i=1}^{m} l^*(-u_i) + r\mathbf{1}^\top \boldsymbol{s} \tag{5.21a}$$

$$\text{s.t. } \boldsymbol{u}^\top \operatorname{diag}(\boldsymbol{y}) H \preccurlyeq \boldsymbol{s}^\top, \tag{5.21b}$$

$$\boldsymbol{s} \succcurlyeq \mathbf{0}. \tag{5.21c}$$

Note that here we have reversed the sign of $\boldsymbol{u}$ too. Essentially the above dual problem can be converted into the following unconstrained problem

$$\min_{\boldsymbol{u}} \sum_{i=1}^{m} l^*(-u_i) + r \sum_{j=1}^{n} \left[\sum_{i=1}^{m} u_i y_i H_{ij} \right]_+ , \tag{5.22}$$

where $[z]_+ = \max(0, z)$ ($z = \sum_{i=1}^{m} u_i y_i H_{ij}$) is the hinge loss. That means, if the edge of a weak classifier is non-positive, it does not have any impact on the optimization problem. The first term can be seen as a regularization term that makes the sample weight $\boldsymbol{u}$ uniform and the second term encourages the edge of a weak classifier to become non-positive. Let us define a symbol, the nonnegatively clipped edge of weak classifier j,

$$d_j^+ = \left[\sum_{i=1}^{m} u_i y_i H_{ij} \right]_+ \tag{5.23}$$

for convenience. As we will see, with the notation of this nonnegatively clipped edge, we are able to unify the Lagrange dual formulations of ℓ_p ($p = 1, 2, \infty$) regularized boosting. So (5.22) is

$$\min_{\boldsymbol{u}} \sum_{i=1}^{m} l^*(-u_i) + r \left\| \boldsymbol{d}^+ \right\|_1 , \tag{5.24}$$

with $\boldsymbol{d}^+ = [d_1^+, \cdots, d_j^+, \cdots, d_n^+]^\top$. To establish the connection with the ℓ_1 regularized boosting, we have the following result:

Proposition 5.4.1. *The Lagrange dual of the ℓ_1-norm regularized boosting can be equivalently written as the following unconstrained optimization*

$$\min_{\boldsymbol{u}} \sum_{i=1}^{m} l^*(-u_i) + r \left\| \boldsymbol{d}^+ \right\|_\infty . \tag{5.25}$$

Proof: Let us start from rewriting the primal problem (5.5) of the ℓ_1 norm regularized boosting. Clearly we can also write the regularization as an explicit constraint

$$\min_{\boldsymbol{w}, \boldsymbol{\gamma}} \sum_{i=1}^{m} l(\gamma_i), \text{ s.t. } \|\boldsymbol{w}\|_1 \leq r, \boldsymbol{w} \succcurlyeq \boldsymbol{0}, \gamma_i = y_i H_{i:} \boldsymbol{w}, \forall i. \tag{5.26}$$

Given the regularization constant ν in (5.5), one can always find a r such that (5.5) and (5.26) have the same solution. It is easy to see that the optimal $\boldsymbol{w}^*$ always locates at the boundary of the feasibility set: $\|\boldsymbol{w}^*\|_1 = r$ [80]. We use the inequality constraint here. The Lagrange dual of (5.26) is

$$\min_{\boldsymbol{u}, s} \sum_{i=1}^{m} l^*(-u_i) + rs, \text{ s.t. } \boldsymbol{u}^\top \operatorname{diag}(\boldsymbol{y}) H \preccurlyeq s \boldsymbol{1}^\top, s \geq 0. \tag{5.27}$$

Here the dual variables are $\boldsymbol{u}$ and $s \geq 0$. From the first constraint, it is clear that

$$s = \max_{j=1\ldots n} \left\{ \sum_{i=1}^{m} u_i y_i H_{ij} \right\} = \max_{j=1\ldots n} \{d_j\},$$

if the largest edge $\max_j \{d_j\} \geq 0$. Here $d_j = \sum_{i=1}^{m} u_i y_i H_{ij}$ is the edge of weak classifier j. Otherwise $s = 0$ must hold because of the constraint $s \geq 0$. Hence, using the concept of clipped edges (5.23), we have

$$s = \max_j \left\{ d_j^+ \right\} = \|\boldsymbol{d}^+\|_\infty.$$

Now we can eliminate s and rewrite the above problem into (5.25). □

Comparing (5.24) and (5.25), the only difference is the norm employed in the second term. This is not a surprising result if one is aware of ℓ_1 norm and ℓ_∞ norm being dual to each other. We also know that the concept of *margin* that is associated with a sample and the concept of *edge* associated with a weak classifier are dual to each other in boosting.

From the KKT conditions for the ℓ_∞-norm regularized boosting, we have the following equalities at optimality:

$$u_i = -l'(\gamma_i), \forall i = 1 \cdots m, \tag{5.28}$$

which is the same as (5.9). From the complementary conditions, we also have

$$\boldsymbol{s}^\top (\boldsymbol{w} - r\mathbf{1}) = 0;$$
$$\boldsymbol{q}^\top \boldsymbol{w} = 0.$$

Therefore, if $w_j \neq r$, then $s_j = 0$ must hold. If $w_j = r$, then $q_j = 0$; hence $s_j - \sum_{i=1}^{m} u_i y_i H_{ij} = q_j = 0$. In summary, we can obtain both the dual variables $\boldsymbol{u}$ and $\boldsymbol{s}$ from the primal variables $\boldsymbol{w}$ using the following relationships

$$s_j = \begin{cases} 0 & \text{if } w_j < r, \\ \sum_{i=1}^{m} u_i y_i H_{ij} & \text{if } w_j = r; \end{cases} \tag{5.29}$$

and (5.28).

Although the Lagrange dual problems of ℓ_1-norm and ℓ_∞-norm regularized boosting can be rewritten into unconstrained problems, both of them are not differentiable, hence still difficult to solve. That is why we solve the primal problems (5.5) and (5.20) instead (as long as the loss function $l(\cdot)$ is convex and differentiable), and use the optimality conditions to recover the dual variables from the primal variables.

However, we know that the ℓ_2 norm is dual to itself and it is differentiable, unlike the ℓ_1 and ℓ_∞ norms. There is hope that the Lagrange dual problem of ℓ_2 norm regularized boosting can be written as an unconstrained differentiable problem so that it is much easier to solve. We discuss this case in the next section.

5.4.1 Arbitrary Regularization

Let us consider the following very general case. Now we not only assume a general loss function, we also assume that the regularization term is any popular regularization function. Concretely, we have the following form,

$$\min_{\boldsymbol{w}} \sum_{i=1}^{m} l(y_i H_{i:}\boldsymbol{w}) + \nu \cdot \Omega(\boldsymbol{w}) \text{ s.t. } \boldsymbol{w} \succcurlyeq 0. \tag{5.30}$$

We rewrite (5.30) into the following equivalent form by introducing another auxiliary variable η:

$$\begin{aligned} &\min_{\boldsymbol{w},\boldsymbol{\gamma},\boldsymbol{\eta}} \sum_{i=1}^{m} l(\gamma_i) + \nu \cdot \Omega(\boldsymbol{\eta}) \\ &\text{s.t. } \gamma_i = y_i H_{i:}\boldsymbol{w}, \forall i = 1 \ldots n, \boldsymbol{\eta} = \boldsymbol{w}, \boldsymbol{w} \succcurlyeq 0. \end{aligned} \tag{5.31}$$

The Lagrangian is

$$\begin{aligned} L = &\sum_{i=1}^{m} l(\gamma_i) + \nu\Omega(\boldsymbol{\eta}) - \boldsymbol{u}^\top(\boldsymbol{\gamma} - \operatorname{diag}(\boldsymbol{y})H\boldsymbol{w}) \\ &- \boldsymbol{s}^\top(\nu\boldsymbol{\eta} - \nu\boldsymbol{w}) - \boldsymbol{p}^\top\boldsymbol{w}, \end{aligned} \tag{5.32}$$

with $\boldsymbol{p} \succcurlyeq 0$.

The Lagrange dual is

$$\begin{aligned} &\max_{\boldsymbol{u},\boldsymbol{s}} -\sum_{i=1}^{m} l^*(u_i) - \nu \cdot \Omega^*(\boldsymbol{s}) \\ &\text{s.t. } \nu\boldsymbol{s}^\top + \boldsymbol{u}^\top \operatorname{diag}(\boldsymbol{y})H \succcurlyeq 0. \end{aligned} \tag{5.33}$$

It is important to introduce the auxiliary variable $\boldsymbol{\eta}$; otherwise we are not able to arrive at this meaningful dual formulation. As before, we reverse the sign of $\boldsymbol{u}$, and we obtain

$$\begin{aligned} &\min_{\boldsymbol{u},\boldsymbol{s}} \sum_{i=1}^{n} l^*(-u_i) + \nu \cdot \Omega^*(\boldsymbol{s}) \\ &\text{s.t. } \boldsymbol{u}^\top \operatorname{diag}(\boldsymbol{y})H \preccurlyeq \nu\boldsymbol{s}^\top. \end{aligned} \tag{5.34}$$

Next we discuss a special case, namely, ℓ_2 regularization. In the case of ℓ_2 regularization, we set $\Omega(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$, and the Fenchel conjugate $\Omega^*(\boldsymbol{s}) = \frac{1}{2}\|\boldsymbol{s}\|_2^2$. So the primal problem is

$$\min_{\boldsymbol{w}} \sum_{i=1}^{m} l(\gamma_i) + \tfrac{1}{2}\nu\|\boldsymbol{w}\|_2^2, \text{ s.t. } \gamma_i = y_i H_{i:}\boldsymbol{w}, \boldsymbol{w} \succcurlyeq 0. \tag{5.35}$$

	Primal	Dual
ℓ_1	$\min \sum_{i=1}^m l(\gamma_i) + \nu \|\boldsymbol{w}\|_1$	$\min \sum_{i=1}^m l^*(-u_i) + r\|\boldsymbol{d}^+\|_\infty$
ℓ_2	$\min \sum_{i=1}^m l(\gamma_i) + \nu \|\boldsymbol{w}\|_2^2$	$\min \sum_{i=1}^m l^*(-u_i) + r\|\boldsymbol{d}^+\|_2^2$
ℓ_∞	$\min \sum_{i=1}^m l(\gamma_i) + \nu \|\boldsymbol{w}\|_\infty$	$\min \sum_{i=1}^m l^*(-u_i) + r\|\boldsymbol{d}^+\|_1$
	$l(\boldsymbol{\gamma})$: loss in primal	$\|\boldsymbol{d}^+\|_q$: loss in dual
	$\|\boldsymbol{w}\|_p$: regularization in primal	$l^*(\boldsymbol{u})$: regularization in dual

Table 5.2: The primal and dual problems of ℓ_p ($p = 1, 2, \infty$) norm regularized boosting algorithms. Samples' margins $\boldsymbol{\gamma}$ and weak classifiers' clipped edges $\boldsymbol{d}^+$ are dual to each other. ℓ_p regularization in primal corresponds to ℓ_q regularization in dual with $1/p + 1/q = 1$. Note that $\boldsymbol{\gamma}$ is a function of $\boldsymbol{w}$ and $\boldsymbol{d}^+$ is a function of $\boldsymbol{u}$.

The Lagrange dual can be written into an unconstrained problem again:

$$\min_{\boldsymbol{u}} \sum_{i=1}^{m} l^*(-u_i) + r \left\|\boldsymbol{d}^+\right\|_2^2, \tag{5.36}$$

with $r = 0.5/\nu$.

Result 5.4.1. *The cost function in* (5.36) *is differentiable everywhere. Hence gradient descent methods like L-BFGS can be used.*

This result follows the fact that the squared hinge loss is differentiable. So we have answered the conjecture in the last section: we indeed obtain a convex, differentiable unconstrained dual problem for the ℓ_2 norm regularized boosting problem. Note that the Fenchel conjugate $l^*(\cdot)$ may have extra constraints on its variable $\boldsymbol{u}$. For example, in the case of exponential loss, $\boldsymbol{u}$ has non-negativeness constraints, which still can be coped with by L-BFGS-B. In practice, solving the primal problem is still more beneficial because the size of the primal problem is usually smaller than the size of the dual problem ($n < m$).

Table 5.4.1 summarizes our results. Note that it may be possible to extend the analysis to the case of a more general ℓ_p, but it is not of general interest for $p \notin \{1, 2, \infty\}$ in the machine learning community. Moreover, when $p \notin \{1, 2, \infty\}$ the optimization problem becomes much more difficult.

The RKHS regularization term is $\Omega(\boldsymbol{w}) = \boldsymbol{w}^\top K \boldsymbol{w}$ where K is the kernel matrix, usually being strictly positive definite. We can easily show its Lagrange dual by noticing the dual of $\boldsymbol{w}^\top K \boldsymbol{w}$ being $\boldsymbol{w}^\top K^{-1} \boldsymbol{w}$.

5.4.2 How the Fenchel Dual of the Primal Loss Regularizes the Dual Variable

Firstly, we have the following theorem.

Theorem 5.4. *The dual variable $\boldsymbol{u}$ in classification is always a probability distribution up to a normalization factor. This normalization factor has no influence on the AnyBoost$_{\mathrm{TC}}$ algorithms.*

Proof: In classification, the relationship between the margin in the primal and the sample weight in the dual is $u_i = -l'(\gamma_i), \forall i$. This equation holds for all the three ℓ_1, ℓ_2 and ℓ_∞ regularization cases.

We know that typically the classification loss function is convex and monotonically decreasing. Therefore, $u_i = -l'(\gamma_i)$ *must be nonnegative*. This is guaranteed as long as we infer $\boldsymbol{u}$ from the primal. So we do not need explicit constraints to make the sample weights nonnegative if working in the primal.

For some loss functions, such as AdaBoost's log-sum-exp function, $\boldsymbol{u}$ is normalized $\|\boldsymbol{u}\|_1 = 1$. $\boldsymbol{u}$ is a probability distribution on the samples. However, the normalization of $\boldsymbol{u}$ does not have any impact on the algorithm in our framework. It does not affect the selection of weak classifiers or the update of $\boldsymbol{u}$ in the iteration. □

Let us have a close look at the loss function $l(\gamma)$ and its role in the Lagrange dual. From Table 5.4.1, we know that $l^*(-u)$ works as a regularization term. Shen and Li discussed the cases when $l(\cdot)$ is exponential loss, logistic loss and generalized hinge loss, $l^*(\cdot)$ is Shannon entropy, binary entropy and Tsallis entropy, respectively [80]. All of them make the dual variable $\boldsymbol{u}$ *uniform*. In LPBoost that employs the non-differentiable hinge loss, the Fenchel conjugate is an indicator function that caps the dual variable $\boldsymbol{u}$ so that $\boldsymbol{u}$ is confined in a box. $\boldsymbol{u}$ is uniformed in a hard way. Hinge loss is an exception in that it is *not strongly convex* and non-differentiable. It is important because we can view hinge loss as the extreme of many loss functions, *e.g.*, the logistic loss. Theorem 5.5 generalizes the theoretical results of [80].

Theorem 5.5. *Let us assume that $l(\gamma)$ is strictly convex and differentiable everywhere in $(-\infty, +\infty)$. The Fenchel dual $\sum_{i=1}^m l^*(-u_i)$ penalizes the divergence of $\boldsymbol{u}$;* i.e., *$\sum_i l^*(-u_i)$ encourages $\boldsymbol{u}$ become uniform.*

Proof: Because $l(\gamma)$ is strictly convex and differentiable everywhere, we know that the first derivative $l'(\gamma)$ is continuous and monotonically increasing for increasing γ. In this case, the Fenchel duality has the following analytic expression:

$$l^{*\prime}(-u_i) = \gamma_i, \ \forall i. \tag{5.37}$$

Here $l^{*\prime}(\cdot)$ is the first derivative of $l^*(\cdot)$. Because Theorem 5.4 holds for all the three cases considered, $\boldsymbol{u}$ take values in $(0, \kappa)$ with $\kappa > 0$ (κ could be $+\infty$). So $l^*(-u_i)$ is

defined in the domain of $(-\kappa, 0)$. Equations $u_i = -l'(\gamma_i)$ and (5.37) hold at the same time for a pair of $\{u_i, \gamma_i\}, \forall i$. When $\gamma \to +\infty$, $u \to 0$ from left side. With (5.37),

$$l^{*\prime}(u \to 0) \to +\infty. \tag{5.38}$$

When $\gamma \to -\infty$, $u \to -\kappa$, so

$$l^{*\prime}(u \to -\kappa) \to -\infty. \tag{5.39}$$

Therefore, $l^{*\prime}(-u^\circ) = 0$ for a certain $0 < -u^\circ < \kappa$. In other words, $l^*(-u)$ must be "$\cup$-shaped" in $0 < -u < \kappa$.

$l^*(-u)$ is convex and has a unique minimum at $0 < -u^\circ < \kappa$. Clearly, $\sum_{i=1}^m l^*(-u_i)$ penalizes those u_i's that deviate from u°. □

In the above theorem, we have assumed that $l(\cdot)$ has no non-differentiable points; it should not be difficult to extend this to the case that $l(\cdot)$ has non-differentiable points using the definition of Fenchel conjugate.

It has been shown in [80] that minimizing the exponential loss function also results in minimizing the divergence of margins. The authors theoretically proved that AdaBoost (and its totally corrective version) approximately maximizes the unnormalized average margin and at the same time minimizes the variance of the margin distribution under the assumption that the margin follows a Gaussian distribution. They have proved this result by analyzing the *primal* optimization problem. Now with Theorem 5.5, we can show this result from the *dual* problem. With $u = -l'(\gamma)$, minimizing the divergence of $\boldsymbol{u}$ also minimizes the divergence of $l'(\boldsymbol{\gamma})$. But generally it cannot be translated into minimizing the divergence of $\boldsymbol{\gamma}$ *unless* $l'(\cdot)$ *is strictly monotonic*. When the exponential loss is used, $u = -l'(\gamma) = \exp(-\gamma)$, in which $l'(\cdot)$ is indeed strictly monotonic. See Fig. 5.1 for a demonstration of this relationship. For the logistic loss, this conclusion also holds. For those loss functions whose first derivatives are truncated from above, *e.g.*, the MadaBoost loss, this conclusion applies only approximately. For instance, in the case of the MadaBoost loss, as long as the margin γ_i is positive, the corresponding u_i equals a constant.

$\sum_i l^*(-u_i)$ in the dual, which is a *hard* indicator function. Essentially they are a set of box constraints on $\boldsymbol{u}$; The first derivative of the hinge loss is a non-continuous step function. Unlike the exponential or logistic loss, to minimize the divergence of $\boldsymbol{u}$ does not effectively minimize the divergence of the margins. If a small margin divergence does contribute to a better generalization capability, the hinge loss would not be an ideal choice for boosting.

Note that the optimization strategy of boosting is entirely different from support vector machines (SVMs). In SVMs, the hypotheses/dictionary for building the final classifier is fixed. In boosting, the weak hypotheses could be infinitely large. The success of hinge loss in SVMs may not be transfered to boosting. However, more work is needed to verify these conjectures.

5.4.3 Confidence-rated Predictions

The derivations of the last sections do not depend on the condition that the output of the weak classifier (the matrix H) must be discrete $\{-1, +1\}$. Therefore, in the case of LPBoost [16], the proposed methods can use a weak learner belonging to a finite set of confidence-related functions. The outputs of the weak learner can be any real values.

Indeed, it is not difficult to show that the same framework can be applied to learn a mixture of kernels as in [3]. It is also possible to include an offset in the learned strong classifier, in which case the optimization problem is only slightly different. However, this topic is beyond the scope of this work.

5.4.4 AnyBoost$_{\mathrm{TC}}$ for Regression

In this section, we cope with regression problems. The presented framework can be easily extended to regression. The difference is the concept of margin $\gamma = yF(\boldsymbol{x})$. In classification, one tries to push this margin as large as possible. In regression, one tries instead to minimize the distance of predicted response $F(\boldsymbol{x})$ and the given response y, *i.e.*, $\min\, l(F(\boldsymbol{x}) - y)$. Here $l(\cdot)$ is usually a convex loss function that penalizes the deviation between $F(\boldsymbol{x})$ and y. Let us consider the arbitrary regularization. We rewrite the problem in (5.31):

$$\begin{aligned} &\min_{\boldsymbol{w},\boldsymbol{\gamma},\boldsymbol{\eta}} \sum_{i=1}^{m} l(\gamma_i) + \nu \cdot \Omega(\boldsymbol{\eta}) \\ &\text{s.t. } \gamma_i = y_i - H_{i:}\boldsymbol{w}\,(\forall i = 1 \dots n), \boldsymbol{\eta} = \boldsymbol{w},\ \boldsymbol{w} \succcurlyeq 0. \end{aligned} \tag{5.40}$$

Compared with (5.31), the only difference is the first constraint, the definition of margins. Using the same technique, we arrive at the corresponding Lagrange dual:

$$\min_{\boldsymbol{u},\boldsymbol{s}} \sum_{i=1}^{n} l^*(u_i) + \nu\Omega^*(\boldsymbol{s}) - \boldsymbol{y}^\top \boldsymbol{u},\ \text{s.t. } \boldsymbol{u}^\top H \preccurlyeq \nu \boldsymbol{s}^\top. \tag{5.41}$$

The KKT condition is

$$u_i = l'(\gamma_i),\ \forall i = 1 \dots m. \tag{5.42}$$

The following are the special cases of AnyBoost$_{\mathrm{TC}}$ for regularization (primal (5.40) and dual (5.41)).

ℓ_1 regularized primal can be written as

$$\min_{\boldsymbol{w},\boldsymbol{\gamma}} \sum_{i=1}^{m} l(\gamma_i),\ \text{s.t.}\ \|\boldsymbol{w}\|_1 \leq r, \boldsymbol{w} \succcurlyeq \mathbf{0}, \gamma_i = y_i - H_{i:}\boldsymbol{w}\ (\forall i). \tag{5.43}$$

Its corresponding dual is

$$\min_{\boldsymbol{u},s} \sum_{i=1}^{m} l^*(u_i) - \boldsymbol{y}^\top \boldsymbol{u} + rs,\ \text{s.t.}\ \boldsymbol{u}^\top H \preccurlyeq s\mathbf{1}^\top, s \geq 0. \tag{5.44}$$

Similar to (5.23), we define

$$d_j^+ = \left[\sum_{i=1}^{m} u_i H_{ij}\right]_+. \tag{5.45}$$

The dual (5.44) can be written into

$$\min_{\boldsymbol{u}} \textstyle\sum_{i=1}^{m} l^*(u_i) - \boldsymbol{y}^\top \boldsymbol{u} + r\|\boldsymbol{d}^+\|_\infty.$$

ℓ_∞ regularized primal writes

$$\min_{\boldsymbol{w},\boldsymbol{\gamma}} \sum_{i=1}^{m} l(\gamma_i),\ \text{s.t.}\ \|\boldsymbol{w}\|_\infty \leq r, \boldsymbol{w} \succcurlyeq \mathbf{0}, \gamma_i = y_i - H_{i:}\boldsymbol{w}\ (\forall i). \tag{5.46}$$

Its dual is

$$\min_{\boldsymbol{u}} \textstyle\sum_{i=1}^{m} l^*(u_i) - \boldsymbol{y}^\top \boldsymbol{u} + r\|\boldsymbol{d}^+\|_1.$$

ℓ_2 regularized primal writes

$$\min_{\boldsymbol{w},\boldsymbol{\gamma}} \sum_{i=1}^{m} l(\gamma_i) + \tfrac{1}{2}\nu\|\boldsymbol{w}\|_2^2,\ \text{s.t.}\ \boldsymbol{w} \succcurlyeq \mathbf{0}, \gamma_i = y_i - H_{i:}\boldsymbol{w}\ (\forall i). \tag{5.47}$$

The dual is

$$\min_{\boldsymbol{u}} \textstyle\sum_{i=1}^{m} l^*(u_i) - \boldsymbol{y}^\top \boldsymbol{u} + \frac{1}{2\nu}\|\boldsymbol{d}^+\|_2^2.$$

The dual Lagrange multiplier $\boldsymbol{u}$ here could be negative from (5.42) because the regression loss function must not be monotonic. The derivative of the loss function could be negative or positive. Therefore in regression, the dual variable $\boldsymbol{u}$ can be viewed a sample weight that measures the importance of the training sample. Generally the regression loss function $l(\cdot)$ is symmetric, therefore the absolute value of $|\boldsymbol{u}|$ could be seen as the weight associated with the training samples.

This is different from the classification case as we have shown in the last section.

Method	Parameter	Parameter Candidates
adaboost	n	$\{10, 20, 50, 100, 200, 300, 500, 1000\}$
tcaboost_exp_l1	ν	$\{0.07, 0.1, 0.3, 1, 5, 7, 10, 20\}$
tcaboost_exp_l2	ν	$\{0.01, 0.05, 0.1, 0.5, 5, 7, 10, 50\}$
tcaboost_exp_linf	ν	$\{0.001, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5\}$
tcaboost_hin_l1	ν	$\{0.005, 0.01, 0.02, 0.1, 0.2, 0.5, 1, 5\}$
tcaboost_hin_l2	ν	$\{0.0002, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 1\}$
tcaboost_hin_linf	ν	$\{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.02, 0.1, 0.2\}$

Table 5.3: The candidates for n in AdaBoost and the trade-off parameter ν in various classification formulations. Note that the size of every candidate-group is 8.

5.5 Experiments

In this section, we run some experiments to verify the performance and efficiency of AnyBoost$_{\mathrm{TC}}$ with various loss functions and regularization terms. More specifically, exponential loss ($l(y, F) = \exp(-yF)$) and hinge loss ($\max(0, -yF)$) are used as loss function candidates, while the regularization term is either ℓ_1, ℓ_2, or ℓ_∞ norm. Consequently, there are 6 combinations of the loss function plus the regularization term. In order to control the complexity of weak classifiers, we have used decision stumps. AnyBoost$_{\mathrm{TC}}$ based on all the loss-regularizer pairs are implemented. AdaBoost is also compared as the baseline. A 5-fold cross-validation procedure is used for each AnyBoost$_{\mathrm{TC}}$ to tune the regularization parameter. For fair comparison, the stopping iteration for AdaBoost is also cross-validated. A collection of parameter candidates for each of the boosting algorithm is shown in Table 5.3.

The first experiment is carried out on the 13 UCI datasets obtained from [62]. Each dataset is randomly split into two groups. 60% of data samples are used for training and validation and the remaining ones are used for test[‡]. L-BFGS-B is used to solve the primal optimization problem with the exponential loss, while Mosek [56] is used to solve the ones with hinge loss, which are not differentiable. The convergence threshold ε is 10^{-5} for the exponential loss case and 10^{-3} for the ones with hinge loss because Mosek is much slower than L-BFGS-B. All the experiments are repeated 10 times and both the mean and standard deviations are reported. Both test and training errors of boosting algorithms are reported in Table 5.4 and 5.5. In the case that an algorithm converges earlier than any preselected iterations, we simply copy the converged results to this iteration and the latter ones.

As demonstrated in the table, the combination "exponential with ℓ_2" achieves the

[‡]For the datasets ringnorm, twonorm and waveform, only 10% of data samples are selected for training and validation due to the extreme large amount of samples.

lowest *training error* on 8 datasets over total thirteen ones while AdaBoost and "exponential with ℓ_∞" only obtain the best performance once. In terms of the *test error*, which is of more interest, on the contrary, "exponential with ℓ_∞" ranks first on 5 datasets comparing with 2 or 3 for other AnyBoost$_{\mathrm{TC}}$. Standard AdaBoost only wins once, which is almost the worst performance among all the algorithms. Considering that cross-validation is also performed for AdaBoost, we may draw the conclusion that in general, AnyBoost$_{\mathrm{TC}}$ could be slightly better if the regularization is carefully selected.

In order to verify the performances of AnyBoost$_{\mathrm{TC}}$ statistically, we implement the Wilcoxon signed-rank test on the experimental results. The Wilcoxon signed-ranks test (WSRT) [18] is a non-parametric alternative of the paired t-test, which can rank the difference in performance of two classifiers for each data set. In this paper, the WSRT test is used for comparing all the boosting algorithms pair-wisely in terms of empirical test error. The null-hypothesis declares that the concerning algorithm is not better than the other method in terms of performance. Thus, it is a one-tail test. We set conventional confidence level to be 95% and the rejection region is $\{w \in \mathbb{R} \mid w > 70\}$, considering the number of datasets with different performances is 13.[§] The output of WSRT is illustrated in Table 5.6.

We can see that, statistically, 1) AnyBoost$_{\mathrm{TC}}$ with exponential loss and ℓ_∞ is superior to AdaBoost; 2) "hinge with ℓ_2" is better than "hinge with ℓ_1" and "hinge with ℓ_∞". This hypothesis test also suggests that AdaBoost is never significantly better than any form of AnyBoost$_{\mathrm{TC}}$.

We have also performed another statistical test, namely the Bonferroni-Dunn test [18]. The result shows that no algorithm statistically outperforms the other one. The comparison results are in Table 5.5. Note that the Bonferroni-Dunn test is a post-hoc manner to verify whether a classifier over-performs the others under the circumstance of multiple comparison [18]. It is not a pair-wise comparison.

The computational complexity of AdaBoost is trivial due to the closed-form solution at each iteration. In contrast, many totally corrective boosting methods [16, 80, 97, 96] are computationally demanding because usually complicated convex problems are involved. By explicitly establishing the primal and dual problems, we can solve the primal that has some special structure to exploit. We use L-BFGS-B to solve the primal. Compared with conventional totally corrective algorithms that use standard convex solvers, ours is much faster. Fig. 5.2 illustrates the time consumption of training for AdaBoost, AnyBoost$_{\mathrm{TC}}$ (exponential loss with ℓ_1) that solves the primal

[§]Here the critical value is not fixed since it depends on the number of datasets with different performances of two algorithms.

with L-BFGS-B and that solves the dual with Mosek, respectively. Note that with this loss-regularization combination, AnyBoost$_{\mathrm{TC}}$ solves essentially the same problem as AdaBoost. We can clearly see the advantage of solving the primal.

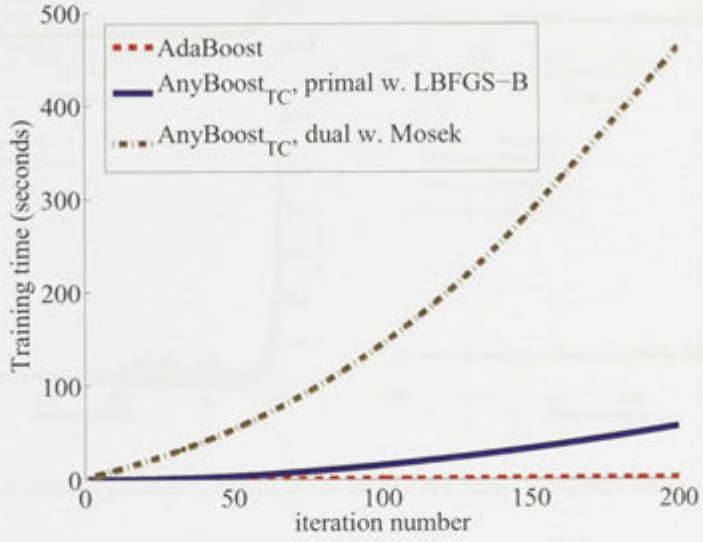


Figure 5.2: Cumulative training time needed for AdaBoost and AnyBoost$_{\mathrm{TC}}$ (exponential loss and ℓ_1 norm regularization) that solves the primal using LBFGS-B and that solves the dual using Mosek. We can see that using LBFGS-B to solve the primal is much faster. Here we have used 80% of the banana dataset for training. A standard desktop PC is used here.

The second experiment is to evaluate the classification performance of different loss functions on the noisy data. Exponential loss is usually very sensitive to the noise owing to the "over-penalty" for the training samples with negative margin (see the loss function shape in Fig. 5.1). In contrast, logistic loss and MadaBoost loss are supposed to be much more robust on noisy data. This experiment is performed on several artificial noisy datasets to confirm this assumption. The noise is generated by randomly flipping the labels of a subset of the original dataset. Our test is carried out with AdaBoost and ℓ_1 norm regularized AnyBoost$_{\mathrm{TC}}$ with logistic loss and MadaBoost loss on 4 noisy datasets that originated from image, ringnorm, twonorm and waveform respectively. 20% of the labels are flipped. Experimental results are displayed in Table 5.8 and the error curves are plotted in Figure 5.3. We can see that, the exponential loss is almost always inferior to the MadaBoost loss and the logistic loss.

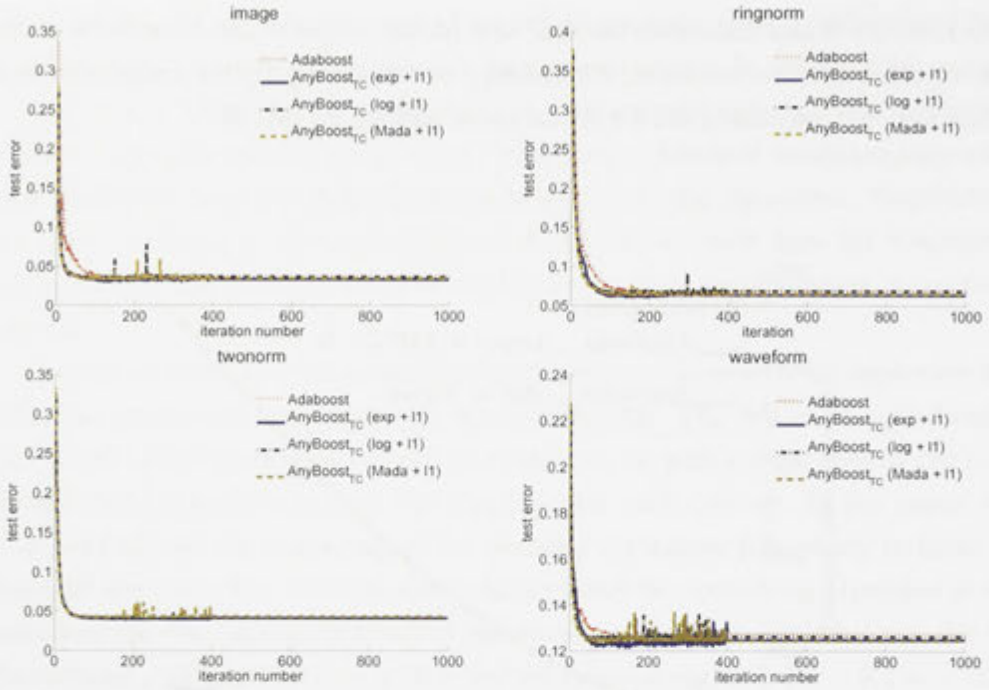


Figure 5.3: Test error curves for AdaBoost and three AnyBoost$_{\mathrm{TC}}$-like algorithms on noisy datasets

5.6 Conclusion and Discussion

In this chapter, we have presented an abstract totally-corretive boosting framework (AnyBoost$_{\mathrm{TC}}$) that can be used to minimize a broad range of regularized risk. An objective in the form of an arbitrary convex loss function plus an arbitrary convex regularization term can be minimized using the boosting technique developed here. We have also shown that a few existing boosting algorithms can be interpreted within our framework.

Like the seminal work of AnyBoost [52], which has been extensively used to design new stage-wise boosting methods, the proposed AnyBoost$_{\mathrm{TC}}$ framework may inspire new totally corrective boosting algorithms. Compared with stage-wise boosting, totally corrective boosting is more flexible, in the sense that domain knowledge in the format of additional constraints may be taken into consideration. The stage-wise AnyBoost cannot deal with constraints.

Another flexibility of totally corrective boosting is that multiple weak learners can be added into the strong learner. As long as the added weak learners violate the current solution, it is guaranteed that the primal cost will be reduced. In contrast, stage-wise boosting can only include one weak classifier per iteration. Our preliminary exper-

iments on training a face detector show that the training time can be reduced when two weak classifiers are added at each iteration, without compromising the detection performance.

Data	Method	Train 50	Train 100	Train 500	Train 1000	Test 50	Test 100	Test 500	Test 1000
	AdaBoost	26.5 ± 1.1	26.0 ± 0.5	25.2 ± 1.0	25.0 ± 1.2	28.6 ± 0.9	28.1 ± 1.0	27.9 ± 1.0	27.8 ± 1.0
	Any exp, ℓ_1	25.0 ± 0.5	23.9 ± 0.8	$\mathbf{17.7 \pm 2.0}$	$\mathbf{17.7 \pm 2.0}$	27.7 ± 1.4	28.1 ± 1.6	31.6 ± 1.5	31.6 ± 1.5
	Any exp, ℓ_2	24.8 ± 0.8	$\mathbf{22.9 \pm 0.6}$	18.2 ± 0.6	18.2 ± 0.6	28.3 ± 1.4	28.8 ± 0.8	30.5 ± 0.5	30.5 ± 0.5
banana	Any exp, ℓ_∞	28.0 ± 1.2	28.3 ± 1.1	24.3 ± 0.8	24.3 ± 0.8	29.9 ± 1.6	30.3 ± 1.2	27.4 ± 0.8	27.4 ± 0.8
	Any hinge, ℓ_1	$\mathbf{23.0 \pm 0.5}$	22.9 ± 1.0	22.8 ± 1.3	22.8 ± 1.3	$\mathbf{25.8 \pm 1.1}$	$\mathbf{25.9 \pm 0.8}$	26.0 ± 0.8	26.0 ± 0.8
	Any hinge, ℓ_2	26.5 ± 1.5	23.8 ± 1.1	23.3 ± 1.0	23.3 ± 1.0	28.1 ± 1.9	26.1 ± 1.3	$\mathbf{25.7 \pm 0.7}$	$\mathbf{25.7 \pm 0.7}$
	Any hinge, ℓ_∞	33.4 ± 6.2	32.5 ± 6.9	22.2 ± 1.9	22.2 ± 1.9	35.2 ± 6.9	34.7 ± 7.3	26.2 ± 0.6	26.2 ± 0.6
	AdaBoost	22.6 ± 1.6	22.6 ± 1.6	22.6 ± 1.6	22.6 ± 1.6	28.1 ± 2.5	28.1 ± 2.5	28.1 ± 2.5	28.1 ± 2.5
	Any exp, ℓ_1	22.3 ± 1.5	$\mathbf{22.5 \pm 1.8}$	22.6 ± 1.7	22.6 ± 1.7	$\mathbf{26.9 \pm 3.2}$	$\mathbf{26.4 \pm 2.8}$	$\mathbf{27.0 \pm 2.9}$	$\mathbf{27.0 \pm 2.9}$
	Any exp, ℓ_2	$\mathbf{20.9 \pm 1.5}$	22.6 ± 6.2	22.7 ± 4.1	22.7 ± 4.1	28.1 ± 2.0	30.9 ± 7.2	29.2 ± 2.6	29.2 ± 2.6
b-cancer	Any exp, ℓ_∞	23.9 ± 1.4	24.0 ± 1.5	23.0 ± 1.2	23.0 ± 1.2	29.4 ± 4.9	29.0 ± 5.1	29.7 ± 4.2	29.7 ± 4.2
	Any hinge, ℓ_1	22.9 ± 2.9	22.9 ± 2.9	22.9 ± 2.9	22.9 ± 2.9	30.6 ± 4.3	30.6 ± 4.3	30.6 ± 4.3	30.6 ± 4.3
	Any hinge, ℓ_2	24.0 ± 1.9	22.5 ± 2.6	$\mathbf{21.6 \pm 3.7}$	$\mathbf{21.6 \pm 3.7}$	30.7 ± 4.1	30.1 ± 4.7	29.8 ± 5.0	29.8 ± 5.0
	Any hinge, ℓ_∞	24.8 ± 1.8	24.8 ± 1.8	22.9 ± 3.4	22.9 ± 3.4	30.7 ± 4.3	30.7 ± 4.3	30.5 ± 4.3	30.5 ± 4.3
	AdaBoost	19.1 ± 2.1	18.4 ± 2.8	18.0 ± 3.3	18.0 ± 3.3	23.8 ± 1.3	24.2 ± 1.3	24.4 ± 1.4	24.4 ± 1.4
	Any exp, ℓ_1	18.5 ± 2.1	18.6 ± 2.4	18.5 ± 2.8	18.5 ± 2.8	$\mathbf{23.6 \pm 1.2}$	$\mathbf{23.5 \pm 1.2}$	25.6 ± 2.2	25.6 ± 2.2
	Any exp, ℓ_2	$\mathbf{15.6 \pm 2.8}$	$\mathbf{11.8 \pm 4.7}$	$\mathbf{6.5 \pm 5.0}$	$\mathbf{6.5 \pm 5.0}$	24.8 ± 2.6	25.5 ± 3.0	27.8 ± 2.9	27.8 ± 2.9
diabetis	Any exp, ℓ_∞	24.3 ± 1.5	22.4 ± 1.5	19.5 ± 1.3	19.5 ± 1.3	26.3 ± 1.6	25.6 ± 0.9	$\mathbf{23.7 \pm 1.3}$	$\mathbf{23.7 \pm 1.3}$
	Any hinge, ℓ_1	18.2 ± 4.1	17.1 ± 5.0	17.1 ± 5.0	17.1 ± 5.0	25.7 ± 1.3	25.4 ± 1.0	25.4 ± 1.0	25.4 ± 1.0
	Any hinge, ℓ_2	23.8 ± 1.4	23.0 ± 1.8	18.3 ± 3.4	18.3 ± 3.4	25.9 ± 0.6	25.4 ± 1.0	25.4 ± 1.0	25.4 ± 1.0
	Any hinge, ℓ_∞	24.4 ± 0.5	24.3 ± 0.5	19.4 ± 5.0	19.4 ± 5.0	26.2 ± 0.7	26.2 ± 0.7	25.7 ± 0.9	25.7 ± 0.9
	AdaBoost	28.3 ± 3.6	28.1 ± 3.8	28.1 ± 3.8	28.1 ± 3.8	39.3 ± 6.1	38.8 ± 6.0	38.8 ± 6.0	38.8 ± 6.0
	Any exp, ℓ_1	28.3 ± 2.8	28.4 ± 2.8	28.6 ± 2.9	28.6 ± 2.9	37.9 ± 4.8	37.4 ± 4.8	37.2 ± 4.5	37.2 ± 4.5
	Any exp, ℓ_2	28.8 ± 3.8	28.5 ± 3.3	29.1 ± 2.9	29.1 ± 2.9	$\mathbf{36.9 \pm 3.1}$	$\mathbf{36.0 \pm 4.3}$	36.6 ± 3.2	36.6 ± 3.2
f-solar	Any exp, ℓ_∞	32.2 ± 3.8	30.0 ± 3.8	28.4 ± 4.1	28.4 ± 4.1	41.7 ± 8.8	38.4 ± 6.9	37.9 ± 4.4	37.9 ± 4.4
	Any hinge, ℓ_1	$\mathbf{27.3 \pm 4.5}$	$\mathbf{27.3 \pm 4.5}$	27.3 ± 4.5	27.3 ± 4.5	38.6 ± 4.4	38.6 ± 4.4	38.6 ± 4.4	38.6 ± 4.4
	Any hinge, ℓ_2	30.1 ± 5.5	28.9 ± 6.3	26.9 ± 4.0	26.9 ± 4.0	40.3 ± 6.9	39.4 ± 4.2	$\mathbf{36.6 \pm 2.8}$	$\mathbf{36.6 \pm 2.8}$
	Any hinge, ℓ_∞	33.0 ± 5.5	32.7 ± 5.6	$\mathbf{26.6 \pm 4.4}$	$\mathbf{26.6 \pm 4.4}$	44.8 ± 9.4	46.1 ± 7.9	39.2 ± 3.8	39.2 ± 3.8
	AdaBoost	20.5 ± 1.4	20.2 ± 1.8	20.2 ± 1.8	20.2 ± 1.8	25.5 ± 1.8	25.4 ± 1.8	25.4 ± 1.8	25.4 ± 1.8
	Any exp, ℓ_1	19.2 ± 1.4	18.7 ± 1.7	18.1 ± 2.0	18.1 ± 2.0	$\mathbf{25.2 \pm 1.9}$	$\mathbf{25.3 \pm 2.4}$	25.7 ± 1.4	25.7 ± 1.4
	Any exp, ℓ_2	18.1 ± 1.7	$\mathbf{15.9 \pm 1.3}$	$\mathbf{10.9 \pm 1.6}$	$\mathbf{10.9 \pm 1.6}$	25.9 ± 2.5	26.6 ± 2.3	28.4 ± 2.1	28.4 ± 2.1
german	Any exp, ℓ_∞	26.6 ± 1.1	24.4 ± 1.3	19.8 ± 1.5	19.8 ± 1.5	28.9 ± 2.3	27.5 ± 1.8	$\mathbf{25.0 \pm 1.7}$	$\mathbf{25.0 \pm 1.7}$
	Any hinge, ℓ_1	$\mathbf{17.2 \pm 2.3}$	16.5 ± 3.1	16.4 ± 3.2	16.4 ± 3.2	25.8 ± 2.8	26.3 ± 2.6	26.4 ± 2.7	26.4 ± 2.7
	Any hinge, ℓ_2	25.1 ± 2.5	21.1 ± 1.6	17.4 ± 1.2	17.4 ± 1.2	28.8 ± 2.8	26.7 ± 2.7	25.7 ± 1.9	25.7 ± 1.9
	Any hinge, ℓ_∞	27.5 ± 1.5	27.5 ± 1.5	17.5 ± 1.0	17.5 ± 1.0	30.4 ± 2.1	30.4 ± 2.1	25.7 ± 2.0	25.7 ± 2.0
	AdaBoost	12.3 ± 3.4	11.9 ± 4.2	11.9 ± 4.2	11.9 ± 4.2	18.8 ± 2.7	19.2 ± 3.0	19.2 ± 3.0	19.2 ± 3.0
	Any exp, ℓ_1	11.4 ± 4.4	11.2 ± 4.4	10.9 ± 4.3	10.9 ± 4.3	18.1 ± 3.3	18.4 ± 3.3	18.3 ± 3.3	18.3 ± 3.3
	Any exp, ℓ_2	9.6 ± 3.5	$\mathbf{7.9 \pm 4.0}$	$\mathbf{4.1 \pm 2.9}$	$\mathbf{4.1 \pm 2.9}$	18.5 ± 2.8	19.3 ± 3.4	20.3 ± 3.4	20.3 ± 3.4
heart	Any exp, ℓ_∞	16.9 ± 4.4	13.6 ± 3.4	10.6 ± 5.1	10.6 ± 5.1	20.4 ± 6.4	18.0 ± 4.7	$\mathbf{17.8 \pm 3.3}$	$\mathbf{17.8 \pm 3.3}$
	Any hinge, ℓ_1	$\mathbf{9.6 \pm 1.6}$	9.6 ± 1.6	9.6 ± 1.6	9.6 ± 1.6	$\mathbf{17.9 \pm 2.7}$	$\mathbf{17.9 \pm 2.7}$	17.9 ± 2.7	17.9 ± 2.7
	Any hinge, ℓ_2	17.0 ± 5.9	14.5 ± 6.0	8.6 ± 4.1	8.6 ± 4.1	22.3 ± 4.3	20.5 ± 2.5	19.0 ± 3.2	19.0 ± 3.2
	Any hinge, ℓ_∞	19.3 ± 7.7	16.3 ± 7.7	7.6 ± 7.2	7.6 ± 7.2	24.4 ± 3.0	22.2 ± 3.1	21.1 ± 3.0	21.1 ± 3.0
	AdaBoost	3.8 ± 0.3	2.2 ± 0.2	0.1 ± 0.1	0.1 ± 0.1	5.6 ± 0.6	4.2 ± 0.6	3.2 ± 0.6	3.2 ± 0.6
	Any exp, ℓ_1	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{3.9 \pm 0.6}$	$\mathbf{3.4 \pm 0.6}$	$\mathbf{3.2 \pm 0.5}$	$\mathbf{3.2 \pm 0.5}$
	Any exp, ℓ_2	0.4 ± 0.5	0.2 ± 0.3	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	4.4 ± 0.6	3.6 ± 0.7	3.3 ± 0.6	3.3 ± 0.6
image	Any exp, ℓ_∞	5.8 ± 2.3	2.7 ± 1.2	0.0 ± 0.1	0.0 ± 0.1	8.2 ± 2.9	4.7 ± 1.4	3.2 ± 0.6	3.2 ± 0.6
	Any hinge, ℓ_1	1.3 ± 0.7	0.8 ± 0.7	0.6 ± 0.6	0.6 ± 0.6	4.0 ± 0.3	3.7 ± 0.5	3.6 ± 0.6	3.6 ± 0.6
	Any hinge, ℓ_2	2.3 ± 1.1	1.4 ± 0.8	0.7 ± 0.7	0.7 ± 0.7	4.5 ± 0.8	3.7 ± 0.7	3.4 ± 0.7	3.4 ± 0.7
	Any hinge, ℓ_∞	20.9 ± 7.0	7.8 ± 2.5	1.0 ± 0.6	1.0 ± 0.6	22.5 ± 6.6	9.7 ± 1.8	3.3 ± 0.5	3.3 ± 0.5

Table 5.4: Training and test errors of AdaBoost, AnyBoost$_{\mathrm{TC}}$ with various loss functions and regularizations. All the algorithms are run 10 times.

Data	Method	Train 50	Train 100	Train 500	Train 1000	Test 50	Test 100	Test 500	Test 1000
	AdaBoost	4.8 ± 0.7	1.6 ± 0.5	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	10.3 ± 0.8	7.5 ± 0.3	$\mathbf{5.6 \pm 0.4}$	$\mathbf{5.6 \pm 0.4}$
	Any exp, ℓ_1	$\mathbf{0.3 \pm 0.4}$	0.1 ± 0.2	0.1 ± 0.1	0.1 ± 0.1	10.5 ± 1.7	6.7 ± 1.0	5.9 ± 0.6	5.9 ± 0.6
	Any exp, ℓ_2	0.8 ± 0.8	$\mathbf{0.1 \pm 0.1}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	10.2 ± 1.9	7.4 ± 0.8	$\mathbf{5.6 \pm 0.4}$	$\mathbf{5.6 \pm 0.4}$
ringnorm	Any exp, ℓ_∞	8.1 ± 1.5	4.0 ± 1.1	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	12.9 ± 1.3	9.2 ± 0.7	5.8 ± 0.4	5.8 ± 0.4
	Any hinge, ℓ_1	1.8 ± 0.7	1.0 ± 0.5	0.5 ± 0.6	0.5 ± 0.6	$\mathbf{7.6 \pm 0.4}$	$\mathbf{5.9 \pm 0.4}$	5.7 ± 0.3	5.7 ± 0.3
	Any hinge, ℓ_2	5.8 ± 1.2	2.8 ± 1.2	1.3 ± 0.5	1.3 ± 0.5	10.7 ± 0.7	7.4 ± 0.7	5.6 ± 0.5	5.6 ± 0.5
	Any hinge, ℓ_∞	27.0 ± 9.8	8.7 ± 2.9	0.5 ± 0.5	0.5 ± 0.5	31.1 ± 9.1	13.1 ± 2.8	7.1 ± 0.6	7.1 ± 0.6
	AdaBoost	7.8 ± 0.5	6.8 ± 0.7	6.7 ± 0.9	6.7 ± 0.9	8.9 ± 0.8	8.7 ± 0.6	8.6 ± 0.5	8.6 ± 0.5
	Any exp, ℓ_1	7.0 ± 0.6	6.2 ± 0.7	8.5 ± 6.7	8.5 ± 6.7	8.5 ± 0.7	8.0 ± 0.7	10.3 ± 5.5	10.3 ± 5.5
	Any exp, ℓ_2	6.9 ± 0.5	6.0 ± 0.6	6.5 ± 1.7	6.5 ± 1.7	8.6 ± 0.7	8.6 ± 0.9	9.8 ± 1.1	9.8 ± 1.1
splice	Any exp, ℓ_∞	12.6 ± 2.3	9.5 ± 1.3	6.3 ± 0.9	6.3 ± 0.9	12.3 ± 2.3	9.7 ± 0.8	8.1 ± 0.8	8.1 ± 0.8
	Any hinge, ℓ_1	$\mathbf{6.3 \pm 0.3}$	$\mathbf{5.9 \pm 0.3}$	$\mathbf{5.7 \pm 0.4}$	$\mathbf{5.7 \pm 0.4}$	$\mathbf{7.8 \pm 0.5}$	$\mathbf{7.6 \pm 0.5}$	$\mathbf{7.7 \pm 0.5}$	$\mathbf{7.7 \pm 0.5}$
	Any hinge, ℓ_2	9.4 ± 1.6	8.2 ± 1.3	6.8 ± 1.0	6.8 ± 1.0	9.7 ± 0.9	8.3 ± 0.6	$\mathbf{7.7 \pm 0.5}$	$\mathbf{7.7 \pm 0.5}$
	Any hinge, ℓ_∞	22.7 ± 2.0	21.4 ± 4.9	6.7 ± 0.9	6.7 ± 0.9	21.9 ± 1.6	20.7 ± 4.2	7.7 ± 0.6	7.7 ± 0.6
	AdaBoost	0.4 ± 0.6	0.4 ± 0.6	0.4 ± 0.6	0.4 ± 0.6	$\mathbf{6.2 \pm 1.8}$	6.5 ± 2.0	6.7 ± 2.2	6.5 ± 2.5
	Any exp, ℓ_1	0.2 ± 0.7	0.2 ± 0.7	0.2 ± 0.7	0.2 ± 0.7	6.3 ± 2.5	$\mathbf{6.3 \pm 2.5}$	$\mathbf{6.3 \pm 2.5}$	$\mathbf{6.3 \pm 2.5}$
	Any exp, ℓ_2	0.6 ± 1.3	0.3 ± 0.7	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	7.6 ± 2.9	6.9 ± 2.6	6.5 ± 2.7	6.5 ± 2.7
thyroid	Any exp, ℓ_∞	5.0 ± 5.1	1.6 ± 1.3	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	10.6 ± 4.0	7.2 ± 2.7	6.7 ± 2.5	6.7 ± 2.5
	Any hinge, ℓ_1	$\mathbf{0.1 \pm 0.3}$	$\mathbf{0.1 \pm 0.3}$	0.1 ± 0.3	0.1 ± 0.3	8.4 ± 2.4	8.4 ± 2.4	8.4 ± 2.4	8.4 ± 2.4
	Any hinge, ℓ_2	1.3 ± 0.9	0.2 ± 0.5	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	8.0 ± 2.6	7.4 ± 2.4	7.7 ± 2.8	7.7 ± 2.8
	Any hinge, ℓ_∞	3.3 ± 2.3	1.3 ± 1.3	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	8.4 ± 3.2	8.3 ± 2.6	7.6 ± 2.5	7.6 ± 2.5
	AdaBoost	26.4 ± 25.0	26.4 ± 25.0	26.4 ± 25.0	26.4 ± 25.0	25.0 ± 25.8	25.0 ± 25.8	25.0 ± 25.8	25.0 ± 25.8
	Any exp, ℓ_1	16.4 ± 7.2	16.4 ± 7.2	16.4 ± 7.2	16.4 ± 7.2	21.0 ± 11.4	21.0 ± 11.4	21.0 ± 11.4	21.0 ± 11.4
	Any exp, ℓ_2	17.1 ± 8.0	17.1 ± 8.0	17.1 ± 8.0	17.1 ± 8.0	19.0 ± 9.4	19.0 ± 9.4	19.0 ± 9.4	19.0 ± 9.4
titanic	Any exp, ℓ_∞	17.1 ± 8.0	17.1 ± 8.0	17.1 ± 8.0	17.1 ± 8.0	21.0 ± 11.4	21.0 ± 11.4	21.0 ± 11.4	21.0 ± 11.4
	Any hinge, ℓ_1	$\mathbf{15.2 \pm 7.5}$	$\mathbf{15.2 \pm 7.5}$	$\mathbf{15.2 \pm 7.5}$	$\mathbf{15.2 \pm 7.5}$	$\mathbf{18.8 \pm 10.5}$	$\mathbf{18.8 \pm 10.5}$	$\mathbf{18.8 \pm 10.5}$	$\mathbf{18.8 \pm 10.5}$
	Any hinge, ℓ_2	$\mathbf{15.2 \pm 7.5}$	$\mathbf{15.2 \pm 7.5}$	$\mathbf{15.2 \pm 7.5}$	$\mathbf{15.2 \pm 7.5}$	$\mathbf{18.8 \pm 10.5}$	$\mathbf{18.8 \pm 10.5}$	$\mathbf{18.8 \pm 10.5}$	$\mathbf{18.8 \pm 10.5}$
	Any hinge, ℓ_∞	$\mathbf{15.2 \pm 7.5}$	$\mathbf{15.2 \pm 7.5}$	$\mathbf{15.2 \pm 7.5}$	$\mathbf{15.2 \pm 7.5}$	$\mathbf{18.8 \pm 10.5}$	$\mathbf{18.8 \pm 10.5}$	20.0 ± 10.0	20.0 ± 10.0
	AdaBoost	0.2 ± 0.3	0.1 ± 0.1	0.1 ± 0.1	0.1 ± 0.1	4.5 ± 0.2	4.1 ± 0.2	4.0 ± 0.3	4.0 ± 0.3
	Any exp, ℓ_1	$\mathbf{0.1 \pm 0.1}$	$\mathbf{0.0 \pm 0.0}$	0.0 ± 0.1	0.0 ± 0.1	4.9 ± 0.3	4.2 ± 0.3	4.1 ± 0.3	4.1 ± 0.3
	Any exp, ℓ_2	0.1 ± 0.3	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{4.3 \pm 0.2}$	3.9 ± 0.1	3.9 ± 0.2	3.9 ± 0.2
twonorm	Any exp, ℓ_∞	1.2 ± 1.4	0.4 ± 0.7	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	5.0 ± 0.7	4.3 ± 0.3	$\mathbf{3.8 \pm 0.2}$	$\mathbf{3.8 \pm 0.2}$
	Any hinge, ℓ_1	0.7 ± 0.4	0.3 ± 0.2	0.3 ± 0.3	0.3 ± 0.3	4.6 ± 0.4	4.2 ± 0.3	4.2 ± 0.3	4.2 ± 0.3
	Any hinge, ℓ_2	1.1 ± 0.8	0.5 ± 0.6	0.2 ± 0.3	0.2 ± 0.3	4.6 ± 0.5	$\mathbf{3.8 \pm 0.2}$	3.9 ± 0.3	3.9 ± 0.3
	Any hinge, ℓ_∞	5.6 ± 5.6	1.1 ± 1.4	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	9.5 ± 5.9	4.5 ± 0.5	4.1 ± 0.4	4.1 ± 0.4
	AdaBoost	3.6 ± 0.9	1.5 ± 1.8	1.3 ± 1.9	1.3 ± 1.9	13.3 ± 0.5	12.9 ± 0.5	13.1 ± 0.5	13.1 ± 0.5
	Any exp, ℓ_1	2.3 ± 1.4	1.9 ± 1.2	1.9 ± 1.2	1.9 ± 1.2	$\mathbf{13.0 \pm 0.7}$	$\mathbf{12.7 \pm 0.4}$	12.8 ± 0.6	12.8 ± 0.6
	Any exp, ℓ_2	$\mathbf{2.0 \pm 2.4}$	$\mathbf{1.1 \pm 1.7}$	$\mathbf{0.2 \pm 0.3}$	$\mathbf{0.2 \pm 0.3}$	13.6 ± 1.1	13.0 ± 0.4	13.1 ± 0.3	13.1 ± 0.3
waveform	Any exp, ℓ_∞	10.3 ± 4.1	7.9 ± 3.6	2.9 ± 2.4	2.9 ± 2.4	15.2 ± 1.5	13.8 ± 0.8	$\mathbf{12.6 \pm 0.7}$	$\mathbf{12.6 \pm 0.7}$
	Any hinge, ℓ_1	4.3 ± 0.9	3.4 ± 1.2	3.3 ± 1.3	3.3 ± 1.3	13.1 ± 0.4	12.8 ± 0.6	12.8 ± 0.7	12.8 ± 0.7
	Any hinge, ℓ_2	13.2 ± 7.8	8.4 ± 5.2	4.8 ± 3.0	4.8 ± 3.0	19.3 ± 4.1	14.9 ± 2.0	12.8 ± 0.7	12.8 ± 0.7
	Any hinge, ℓ_∞	14.4 ± 7.6	12.8 ± 8.5	3.0 ± 2.3	3.0 ± 2.3	19.2 ± 4.4	18.8 ± 4.0	13.1 ± 1.3	13.1 ± 1.3

Table 5.5: Training and test errors of AdaBoost, AnyBoost$_{\mathrm{TC}}$ with various loss functions and regularizations (continuing). All the algorithms are run 10 times.

	AdaBoost	Any exp, ℓ_1	Any exp, ℓ_2	Any exp, ℓ_∞	Any hinge, ℓ_1	Any hinge, ℓ_2	Any hinge, ℓ_∞
AdaBoost	–	No ($46 < 70$)	No ($53 < 61$)	No ($19 < 70$)	No ($44 < 70$)	No ($38 < 70$)	No ($62 < 70$)
Any exp, ℓ_1	No ($45 < 70$)	–	No ($54 < 70$)	No ($24 < 61$)	No ($44 < 70$)	No ($32 < 70$)	No ($60 < 70$)
Any exp, ℓ_2	No ($25 < 61$)	No ($37 < 70$)	–	No ($30 < 70$)	No ($32 < 70$)	No ($24 < 70$)	No ($45 < 70$)
Any exp, ℓ_∞	**Better** ($72 > 70$)	No ($54 < 61$)	No ($61 < 70$)	–	No ($61 < 70$)	No ($47 < 70$)	**Better** ($70 = 70$)
Any hinge, ℓ_1	No ($47 < 70$)	No ($47 < 70$)	No ($59 < 70$)	No ($30 < 70$)	–	No ($11 < 61$)	No ($64 < 70$)
Any hinge, ℓ_2	No ($53 < 70$)	No ($59 < 70$)	No ($67 < 70$)	No ($44 < 70$)	**Better** ($67 > 61$)	–	**Better** ($82 > 70$)
Any hinge, ℓ_∞	No ($29 < 70$)	No ($31 < 70$)	No ($46 < 70$)	No ($21 < 70$)	No ($27 < 70$)	No ($9 < 70$)	–

Table 5.6: Results of the Wilcoxon Signed-Ranks Test (WSRT) [18]. The test is processed pairwisely among the compared boosting methods. The block where "Better" takes place indicates that the algorithm corresponding to its row is better than the algorithm corresponding to its column. "No" suggests that the row algorithm is not better than the column algorithm. The inequality in the parenthesis is the comparison between the Wilcoxon statistic (l.h.s.) and the critical value (r.h.s.). The critical value depends on the number of data sets yielding different performances. Hence it is not fixed. The hypothesis is rejected when the statistic is larger than the critical value.

	AdaBoost	Any exp, ℓ_1	Any exp, ℓ_2	Any exp, ℓ_∞	Any hinge, ℓ_1	Any hinge, ℓ_2	Any hinge, ℓ_∞
AdaBoost	–	No ($0.453 < 2.693$)	No ($0.726 < 2.693$)	No ($-1.180 < 2.693$)	No ($1.407 < 2.693$)	No ($-0.771 < 2.693$)	No ($1.497 < 2.693$)
Any exp, ℓ_1	No ($-0.453 < 2.693$)	–	No ($0.226 < 2.693$)	No ($-1.361 < 2.693$)	No ($1.044 < 2.693$)	No ($-1.225 < 2.693$)	No ($0.862 < 2.693$)
Any exp, ℓ_2	No ($-0.726 < 2.693$)	No ($-0.226 < 2.693$)	–	No ($-1.588 < 2.693$)	No ($0.817 < 2.693$)	No ($-1.452 < 2.693$)	No ($0.635 < 2.693$)
Any exp, ℓ_∞	No ($1.180 < 2.693$)	No ($1.361 < 2.693$)	No ($1.588 < 2.693$)	–	No ($2.405 < 2.693$)	No ($0.136 < 2.693$)	No ($2.224 < 2.693$)
Any hinge, ℓ_1	No ($-1.407 < 2.693$)	No ($-1.044 < 2.693$)	No ($-0.817 < 2.693$)	No ($-2.405 < 2.693$)	–	No ($-2.269 < 2.693$)	No ($-0.181 < 2.693$)
Any hinge, ℓ_2	No ($0.771 < 2.693$)	No ($1.225 < 2.693$)	No ($1.452 < 2.693$)	No ($-0.136 < 2.693$)	No ($2.269 < 2.693$)	–	No ($2.088 < 2.693$)
Any hinge, ℓ_∞	No ($-1.497 < 2.693$)	No ($-0.862 < 2.693$)	No ($-0.635 < 2.693$)	No ($-2.224 < 2.693$)	No ($0.181 < 2.693$)	No ($-2.088 < 2.693$)	–

Table 5.7: Results of the Bonferroni-Dunn Test (BDT) [18]. Each algorithm is compared with other 6 boosting methods. The block where "Better" takes place indicates that the algorithm corresponding to its row is better than the algorithm corresponding to its column. "No" means the row algorithm cannot be considered better than the column algorithm. The inequality in the parenthesis is the comparison between the Bonferroni statistic (l.h.s.) and the critical value (r.h.s.). The critical value depends on the number of comparing classifiers, thus it is fixed (here it is 2.693). The hypothesis is rejected when the statistic is larger than the critical value.

		train_10	train_50	train_100	train_500	train_1000	test_10	test_50	test_100	test_500	test_1000
image	adaboost	28.8 ± 1.8	22.9 ± 0.5	21.6 ± 0.8	20.8 ± 1.2	20.8 ± 1.2	19.2 ± 3.2	9.8 ± 1.6	9.6 ± 1.5	10.0 ± 1.5	10.0 ± 1.5
	tcaboost_exp_l1	25.6 ± 0.5	$\mathbf{20.5 \pm 0.2}$	$\mathbf{17.2 \pm 1.1}$	$\mathbf{8.9 \pm 3.8}$	$\mathbf{8.9 \pm 3.8}$	12.9 ± 1.1	13.6 ± 2.5	15.8 ± 3.6	19.6 ± 2.2	19.6 ± 2.2
	tcaboost_lexp_l1	25.1 ± 0.6	20.9 ± 0.4	20.0 ± 0.5	19.7 ± 0.7	19.7 ± 0.7	12.2 ± 0.9	$\mathbf{8.3 \pm 0.6}$	$\mathbf{8.2 \pm 1.3}$	$\mathbf{8.9 \pm 1.8}$	$\mathbf{8.9 \pm 1.8}$
	tcaboost_mada_l1	$\mathbf{24.2 \pm 0.7}$	20.9 ± 0.8	21.4 ± 1.9	21.2 ± 2.1	21.2 ± 2.1	$\mathbf{10.7 \pm 2.0}$	8.5 ± 1.2	9.4 ± 1.7	10.7 ± 1.7	10.7 ± 1.7
ringnorm	adaboost	29.5 ± 1.7	19.6 ± 0.9	17.4 ± 1.5	13.5 ± 5.0	13.0 ± 5.7	27.8 ± 1.0	17.4 ± 1.1	17.3 ± 1.1	17.8 ± 1.2	17.8 ± 1.2
	tcaboost_exp_l1	27.3 ± 0.8	17.7 ± 0.8	$\mathbf{10.2 \pm 3.0}$	$\mathbf{4.3 \pm 3.5}$	$\mathbf{4.3 \pm 3.5}$	$\mathbf{23.4 \pm 0.8}$	19.9 ± 1.3	21.8 ± 2.2	21.6 ± 1.1	21.6 ± 1.1
	tcaboost_lexp_l1	26.9 ± 1.1	17.8 ± 1.0	16.1 ± 1.1	15.7 ± 1.7	15.7 ± 1.7	23.9 ± 1.5	15.2 ± 1.1	14.7 ± 1.3	16.2 ± 1.9	16.2 ± 1.9
	tcaboost_mada_l1	$\mathbf{26.4 \pm 1.2}$	$\mathbf{17.1 \pm 1.8}$	15.6 ± 3.1	14.7 ± 3.4	14.7 ± 3.4	23.6 ± 1.4	$\mathbf{14.6 \pm 1.4}$	$\mathbf{14.3 \pm 2.3}$	$\mathbf{14.6 \pm 3.3}$	$\mathbf{14.6 \pm 3.3}$
twonorm	adaboost	24.5 ± 1.2	20.8 ± 1.8	20.1 ± 2.7	19.7 ± 3.3	19.7 ± 3.3	16.2 ± 0.6	13.2 ± 0.7	13.6 ± 1.0	13.7 ± 1.2	13.7 ± 1.2
	tcaboost_exp_l1	23.6 ± 1.2	$\mathbf{18.7 \pm 1.2}$	$\mathbf{15.3 \pm 3.7}$	$\mathbf{11.6 \pm 4.6}$	$\mathbf{11.6 \pm 4.6}$	15.0 ± 0.4	13.5 ± 3.9	14.5 ± 4.6	22.9 ± 4.6	22.9 ± 4.6
	tcaboost_lexp_l1	$\mathbf{22.9 \pm 1.1}$	19.7 ± 1.0	19.5 ± 0.8	19.5 ± 0.8	19.5 ± 0.8	14.8 ± 0.6	7.6 ± 0.5	7.6 ± 0.5	$\mathbf{7.8 \pm 0.6}$	$\mathbf{7.8 \pm 0.6}$
	tcaboost_mada_l1	23.0 ± 1.1	18.8 ± 0.5	18.8 ± 0.5	18.6 ± 0.5	18.6 ± 0.5	$\mathbf{14.4 \pm 0.4}$	$\mathbf{6.9 \pm 0.5}$	$\mathbf{6.8 \pm 0.5}$	7.9 ± 1.4	7.9 ± 1.4
waveform	adaboost	26.3 ± 1.6	23.8 ± 4.2	23.8 ± 4.2	23.8 ± 4.2	23.8 ± 4.2	20.1 ± 1.6	19.5 ± 1.3	19.5 ± 1.3	19.5 ± 1.3	19.5 ± 1.3
	tcaboost_exp_l1	26.0 ± 1.4	22.9 ± 3.9	22.5 ± 5.7	$\mathbf{19.7 \pm 6.1}$	$\mathbf{19.7 \pm 6.1}$	17.3 ± 1.0	17.7 ± 2.5	18.0 ± 2.5	26.0 ± 5.7	26.0 ± 5.7
	tcaboost_lexp_l1	25.0 ± 1.1	23.2 ± 1.7	22.8 ± 1.9	23.0 ± 1.9	23.0 ± 1.9	$\mathbf{16.2 \pm 1.3}$	$\mathbf{15.2 \pm 0.9}$	$\mathbf{15.2 \pm 0.8}$	$\mathbf{15.3 \pm 0.9}$	$\mathbf{15.3 \pm 0.9}$
	tcaboost_mada_l1	$\mathbf{24.4 \pm 1.7}$	$\mathbf{22.3 \pm 4.0}$	$\mathbf{22.0 \pm 3.9}$	22.7 ± 3.6	22.7 ± 3.6	16.3 ± 1.0	15.6 ± 0.7	15.8 ± 0.9	18.2 ± 2.7	18.2 ± 2.7

Table 5.8: Test on the noisy data: test and training errors of AdaBoost, AnyBoost$_{\mathrm{TC}}$-like algorithms with three types of loss functions (exponential loss, logistic loss and MadaBoost loss) and ℓ_1-norm regularization. The errors are shown in percentage. All the algorithms are run 5 times.

Chapter 6

Boosting the Locality for Robust Face Recognition

6.1 Introduction

Face Recognition (FR) is a long-standing problem in computer vision. In the past decade, much effort has been devoted to the linear representation (linear subspace) based algorithms such as Nearest Feature Line (NFL) [44], Nearest Feature Subspace (NFS) [13], Sparse Representation Classification (SRC) [98], and the recent Linear Regression Classification (LRC) [57]. In these approaches, one tries to *linearly* represent the test image by a certain set of the labeled training images. Compared with traditional FR approaches, improved performance has been observed. Nevertheless, the cropped faces are usually contaminated by noise and occlusion. If the regression problem is defined over all the pixels, the noise and occlusion can deteriorate the final recognition accuracy. As in many standard machine learning methods, these noisy pixels are outliers and should be discarded.

Several heuristic methods were introduced to address the problem. In particular, the modular approach is used in [98] and [57] for eliminating the adverse impact of continuous occlusion. Significant improvement in accuracy was observed thanks to the partition-and-vote (majority voting) strategy. The drawback of these heuristics is clear: one must know *a priori* the size and location of the occlusion, roughly. Otherwise, it would be extremely difficult to partition the face images.

On the other hand, given a set of training faces for an individual, it is also important to determine how many faces and which faces should be used in the regression problem. In NFL, two faces determined by a exhaustive search form a "feature line". SRC implicitly selects the faces by solving an ℓ_1 norm minimization problem, which is a

relaxation of the original non-convex ℓ_0 minimization problem. LRC uses all the faces available for an individual when solving the linear regression problem. The underlying assumption for the linear regression is that the test face and the training faces reside in a linear manifold. And the pattern of the linear manifold is determined by certain assumptions. How can we ensure a validity of this assumption?

Actually, the assumption of the linear model is sometimes too rudimentary in practice and could be invalid if we use all the training faces, especially when serious occlusions or extreme postures exist.

In this chapter, we propose to use boosting for solving the above two problems, *i.e.* how to find the most reliable face parts and the what are the good basis faces for performing the linear representation. Boosting is usually considered as a sophisticated alternative to voting methods, such as bagging [5] and decision forests [38]. Boosting has been extensively studied in computer vision since Viola and Jones introduced their boosting based face detection system [90]. The proposed boosting based face recognition algorithms, termed *Boosted LRC* (BLRC), are built upon the state-of-the-art linear regression classification of Nassem *et al.* [57]. LRC has been shown to achieve very promising results [57]. In the case of contiguous occlusion, LRC outperforms SRC on the AR face data [57]. The BLRC algorithm is able to combine a few weak classifiers, namely, the locality-constrained LRCs, to achieve significantly better performance.

Several works have used ensemble learning methods for face recognition [94, 12, 48, 101, 95]. Our BLRC differs from these existing works in that the weak classifiers used are completely different. Most boosting based face recognizers build an ensemble of weak classifiers such as kNN or Linear Discriminant Analysis (LDA). Little work has been done to select relevant pixels/patches for robust recognition. Our first algorithm, boosted patches with LRC, is largely inspired by Viola and Jones' face detector [90] in the sense that both algorithms select discriminative image patches and train weak classifiers on the image patches. Note that because the final purpose is entirely different, the weak classifiers as well as the boosting algorithms are different—we have used multi-class boosting with LRC. To our knowledge, the second proposed algorithm, boosting locality (neighborhoods of the test face) with LRC, is the first principled discriminative method that addresses the training faces selection for regression based face recognition.

Compared with these existing methods, our approach possesses the following advantages.

1. *Superior functional space based on locality.* In most of the proposed ensemble FR approaches, random subspace or random sampling are adopted for generat-

ing weak learners. However, we argue that one could not exploit the random functional space effectively, especially with a small number of weak learners. In this work, a locality-constraint is enforced on the functional space. The locality here could stands for either the facial patches or the neighborhoods in the feature space. For the first time, we use LRC as weak classifiers. With these two types of localities, we select relevant patches and neighborhoods to overcome the problems of continuous occlusion and severe misalignment/postures, respectively. The excellence of the two constraints are both verified in the experiments.

2. *More elegant ensemble frameworks.* Most of the existing ensemble methods employ majority voting [94, 12, 95] or heuristically modified AdaBoost [48, 95] as their aggregating strategy. In contrast, our method is facilitated with the algorithm termed SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss function) [105], which is a recently proposed multiple-class boosting approach. It can directly boost the multiple-class weak learner and thus perfectly fits LRC. Furthermore, a customized SAMME algorithm is proposed to boost the LRC, based on neighborhoods. Thanks to SAMME and its modified cousin, the BLRC algorithm illustrates higher accuracy and other merits, as we will show later.

   In addition, the totally corrective version of SAMME [37], termed SAMME-CG, is also employed to replace the gradient-descent prototype. Even better performance is observed. Considering the special characteristic of our problem: the amount of weak classifier is finite and relatively small, we aggressively discard the iterative procedure and solve the optimization problem in a single-step over all the weak classifiers. With this simple paradigm, best accuracy is achieved.

3. *Enhanced robustness.* All the previous algorithms only employ relatively simple weak-learners, therefore leading to low robustness towards illumination and other difficulties. LRC, on the other hand, is much more robust to the handicaps. We further boost the performance of LRC under advanced boosting frameworks and some record-breaking recognition rates are observed (see Section 6.6).

4. *Higher efficiency.* The use of LRC as the weak learner has an interesting characteristic: *weak classifiers need to be trained only once* because the prediction of an LRC is independent of the training data's weight distribution. This inspiring characteristic makes the training procedure extremely fast. We also speed up the test process of BLRC, enabling real-time applications.

Moreover, the leave-one-out strategy is used to tailor the instance-based weak classifier to the boosting framework. This strategy brings a unexpected advantage that there is no need to perform a extra validation process to tune the model parameter. Given a parameter, only one training procedure is required. Compared with the n times training in cross-validation, the training speed of the parametric BLRC is significantly increased.

In short, we boost the performance of LRC and achieve best reported recognition rates on standard face recognition benchmark datasets (see Section 6.6), by paying only a marginal price.

The remaining part of this chapter is organized as follows. Before presenting the main algorithms, some preliminaries are introduced in Section 6.2. The main idea about boosting the local face-patches is presented in Section 6.3. We introduce the BLRC-N algorithm, which boosts the local neighborhoods for LRC in Section 6.4. In Section 6.5, two variations of SAMME, the Column-Generation (CG) based SAMME and the single-step SAMME, are adopted for combining weak classifiers. The experiment results and discussions are presented in Section 6.6 and Section 6.7 respectively.

6.2 Preliminaries

In this section, we review some background before we present our main algorithms.

6.2.1 Linear regression classification

For a typical FR problem, one is usually given N vectorized face images $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{d \times N}$ belonging to K different individuals. Let $\mathbf{x}_i^k \in \mathbb{R}^d$ denote the ith face from the kth subject and $\mathbf{X}_k \in \mathbb{R}^{d \times N_k}$ indicate the image collection of the kth class*. LRC solves a set of least squares problems [57]:

$$\min_{\boldsymbol{\beta}_k} \|\mathbf{y} - \mathbf{X}_k \boldsymbol{\beta}_k\|^2 \quad \forall\, k = 1, 2, \ldots, K, \tag{6.1}$$

where $\mathbf{y} \in \mathbb{R}^d$ is the query image. This minimization problem can be solved in closed-form:

$$\beta_k^\star = (\mathbf{X}_k{}^\top \mathbf{X}_k)^{-1} \mathbf{X}_k{}^\top \mathbf{y}; \tag{6.2}$$

then we arrive at the reconstruction residual for class k

$$r_k = \left\| (\mathbf{X}_k (\mathbf{X}_k{}^\top \mathbf{X}_k)^{-1} \mathbf{X}_k{}^\top - \mathbf{I}) \mathbf{y} \right\|^2, \tag{6.3}$$

*Without loss of generality, in this work, we assume all the classes share the same sample number N_k.

where $\mathbf{I} \in \mathbb{R}^{d \times d}$ is the identity matrix. Finally, LRC classifies the query image $\mathbf{y}$ as

$$l^*(y) = \underset{k}{\operatorname{argmin}} \{r_k\}. \tag{6.4}$$

LRC achieves comparable performance with those of state-of-the-art approaches, albeit with the extremely simple strategy. We attempt to further improve the accuracy of LRC.

6.2.2 Multi-class boosting: SAMME

Boosting algorithms are originally designed for binary classification but could be extended for multi-class problems in a few ways [72, 36, 73]. Compared with most conventional multi-class boosting approaches, SAMME [105] requires only that the accuracy of each weak classifier to be better than random guessing, which is $1/K$ (rather than $1/2$ for many multi-class boosting algorithms). Thus it is expected to exploit the functional space of weak classifier more effectively and more flexible. Suppose that we are given a set of training data $(\mathbf{x}_1, l_1), \cdots, (\mathbf{x}_N, l_N)$. where $\mathbf{x}_i$ is the feature vector and $l_i \in \{1, 2, \cdots, K\}$ is the corresponding label. Let $[\![\cdot]\!] \in \{0, 1\}$ denote the boolean operator and ω_i the weight of the ith training data.

Coding strategies Most multiple-class boosting methods involve various coding strategies [72, 36] that encode each label into a vector. The objective is to quantitatively measure the loss between the label and its predictions, for multiple-class problems. The simplest coding strategy is the one-to-one mapping. Given a sample's label l, its mapped code $\mathbf{l} = [\mathbf{l}_1, \mathbf{l}_2, \cdots, \mathbf{l}_K]^\top$ is generated as

$$\mathbf{l}_i = \begin{cases} 1 & i = l, \\ -1 & \text{otherwise;} \end{cases} \tag{6.5}$$

That is to say, only lth element of the vector is $+1$ and others are all -1. SAMME adopts a more advanced coding strategy, which writes

$$\mathbf{l}_i = \begin{cases} 1 & i = l, \\ -\frac{1}{K-1} & \text{otherwise.} \end{cases} \tag{6.6}$$

Note that the coding strategy is enforced not only on the data labels, but also on the outputs of classifiers. Here we denote the mapped output of a weak classifier as $\mathbf{h}(\mathbf{x}) = [\mathbf{h}^1(\mathbf{x}), \mathbf{h}^2(\mathbf{x}), \cdots, \mathbf{h}^K(\mathbf{x})] \in \mathbb{R}^K$, the elements are evaluated in the same way as 6.6. Accordingly, we know that

$$h(\mathbf{x}) = \underset{i}{\operatorname{argmin}}(\mathbf{h}^i(\mathbf{x})),$$

where $h(\mathbf{x})$ is the scalar prediction of $\mathbf{x}$'s label.

SAMME algorithm The procedure of SAMME is summarized in Algorithm 6. For more details of SAMME, please refer to Zhu *et al.*'s paper [105].

Algorithm 6 SAMME: multi-class boosting

Input:

- A training data-label set $(\mathbf{x}_1, l_1), \cdots, (\mathbf{x}_N, l_N)$.
- A data weight set $\omega_1, \cdots, \omega_N$.
- A functional space $\mathbb{H}$ of the weak classifier $h(\mathbf{x})$.
- A maximum iteration T.

begin

· Initialize the training sample weights: $\omega_i = 1/N, \ \ i = 1, 2, \cdots, N$;

for $t \leftarrow 1$ **to** T **do**

· Find the weak classifier $h_t(\mathbf{x})$, such that $h_t = \underset{h \in \mathbb{H}}{\operatorname{argmin}} \sum_{i=1}^{N} \omega_i \cdot [\![h(\mathbf{x}_i) \neq l_i]\!]$;

· Compute $\varepsilon_t = \sum_{i=1}^{N} \omega_i \cdot [\![h_t(\mathbf{x}_i) \neq l_i]\!] / \sum_{i=1}^{N} \omega_i$;

· Set $\alpha_t = \log\left((1 - \varepsilon_t)/\varepsilon_t\right) + \log(K - 1)$;

· Set $\omega_i \leftarrow \omega_i \cdot \exp\left(\alpha_t \cdot [\![h_t(\mathbf{x}_i) \neq l_i]\!]\right)$;

· Re-normalize ω_i;

· **if** converge, break;

end

Output: The strong classifier: $F(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \sum_{t=1}^{T} \alpha_t \cdot [\![h_t(\mathbf{x}) = k]\!]$.

6.2.3 SAMME with column-generation

Hao *et al.* proposed two totally corrective multiple-class boosting algorithms [37], one of which is based on the loss function of SAMME. The algorithm, termed SAMME-CG here, could be viewed as an extension of our totally corrective method to the multiple-class problem.

The exponential loss function used in SAMME writes

$$\begin{aligned} \text{Loss} &= \sum_{i=1}^{N} \exp\left(-\tfrac{1}{K}\mathbf{y}_i^\top \mathbf{f}(\mathbf{x}_i)\right) \\ &= \sum_{i=1}^{N} \exp\left(-\tfrac{1}{K}\mathbf{y}_i^\top \textstyle\sum_{j=1}^{T} \alpha_j \mathbf{h}_j(\mathbf{x}_i)\right) \end{aligned} \tag{6.7}$$

Thus the optimization problem to be solved is

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \exp\left(-\tfrac{1}{K}\textstyle\sum_j \alpha_j \mathbf{y}_i^\top \mathbf{h}_j(\mathbf{x}_i)\right) \\ &\text{s.t. } \boldsymbol{\alpha} \succeq 0, \|\boldsymbol{\alpha}\|_1 \leq \theta. \end{aligned} \tag{6.8}$$

which is obviously a convex problem. As derived in [37], the Lagrange dual problem of (6.8) is

$$\begin{aligned} &\max_{r,\boldsymbol{\omega}} \; -\theta r - \sum_{i=1}^{N} \omega_i \log \omega_i + \sum_{i=1}^{N} \omega_i \\ &\text{s.t. } \frac{1}{K}\sum_{i=1}^{N} \omega_i \mathbf{y}_i^\top [\mathbf{h}_1(\mathbf{x}_i) \cdots \mathbf{h}_T(\mathbf{x}_i)] \leq r\mathbf{1}^\top. \end{aligned} \tag{6.9}$$

Using the idea analogous to Chapter 3, Hao *et al.* obtained the column-generation based multiple-class boosting algorithm, which is termed SAMME-CG in this thesis. Their algorithm is summarized in Algorithm 7: please note that there are two patterns of a weak classifier, the scalar output $h(\cdot)$ and its mapped vector $\mathbf{h}(\cdot)$.

In this chapter, we mainly use the SAMME and its totally corrective variations to boost the performance of the locality-constrained LRC, although other multi-class boosting might also be applicable.

6.3 Boosting Local Patches for LRC

6.3.1 Basic framework

The modular-based approaches used in [98] and [57] essentially suggest that the linear subspace assumption is more reliable on the local patches and the voting procedure can eliminate the impact of the outliers (usually the occlusion) to a great extent.

In this work, we design an enhanced strategy to exploit the structure information of face images. First of all, n_p *face patches* are generated to form a patch set

Algorithm 7 Totally Corrective SAMME

Input:

- A training data $(\mathbf{x}_i, y_i), i = 1 \dots N$.
- A termination threshold $\epsilon > 0$.
- A regularization coefficient θ.
- A maximum training step T.

begin

· Initialize $\boldsymbol{\alpha} = 0, t = 0, \omega_i = 1/N, \forall i = 1 \dots N$;

for $t \leftarrow 1$ **to** T **do**

· Find a new weak classifier h_t such that $h_t = \underset{h \in \mathbb{H}}{\operatorname{argmax}} \sum_{i=1}^{N} \omega_i \mathbf{y}_i^\top \mathbf{h}(\mathbf{x}_i)$;

· **if** $\frac{1}{K} \sum_{i=1}^{N} \omega_i \mathbf{y}_i^\top \mathbf{h}_t(\mathbf{x}_i) < r + \epsilon$, break;

· Add new constraint to the corresponding dual problem;

· Solve the dual (6.9);

· Calculate the primal $\boldsymbol{\alpha}$ according to the solutions of dual and KKT condition;

end

Output: The strong classifier: $F(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \sum_{t=1}^{T} \alpha_t \cdot [\![h_t(\mathbf{x}) = k]\!]$.

$\mathbb{P} = \{p_1, p_2, \cdots, p_{n_p}\}$. Afterwards, the SAMME algorithm is performed to iteratively select the best LRC predictor among all the candidates, with the minimum weighted classification error.

Let $\gamma_j^{\boldsymbol{\omega}}(\cdot)$, $j \in \{1, 2, \cdots, n_p\}$ denotes the LRC predictor based on the jth patch and the weight vector [†] $\boldsymbol{\omega} \in \mathbb{R}^N$ over all the training samples. At iteration t of SAMME, the index $j^\star$ corresponding to the best LRC predictor is determined by

$$j^\star = \underset{j \in \{1,2,\cdots,n_p\}}{\operatorname{argmin}} \sum_{i=1}^{N} \omega_i \cdot [\![\gamma_j^{\boldsymbol{\omega}}(\mathbf{x}_i) \neq l_i]\!], \tag{6.10}$$

namely, the minimum weighted error. Since LRC is a prototype-based weak learner, we do not need to record the classifier model of the selected LRC. Instead, the index $j^\star$ is recorded. Meanwhile, the weight α_t for the current weak classifier is also calculated. Figure 6.1 demonstrates the training process of the proposed BLRC-P ("-P" stands for patch).

In the prediction stage, the selected patches are cropped according to the learned indexes, and the associated LRC predictors are performed for identifying the probe

[†] *i.e.* the data distribution vector

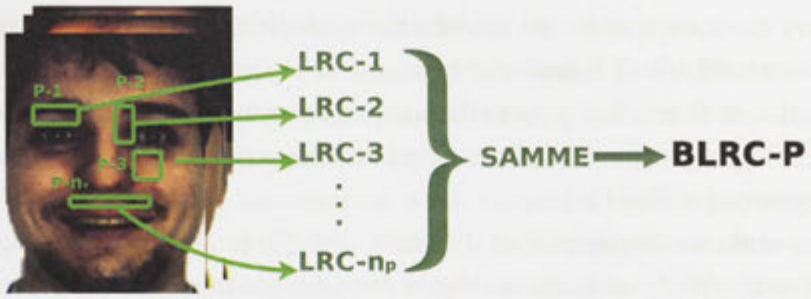


Figure 6.1: Demonstration of the proposed BLRC-P approach. The patches are first cropped out and the LRC's are generated based on these patches. Then, SAMME is used to combine the weak classifiers into a strong one.

face image $\mathbf{y}$, by using the following equation:

$$F(\mathbf{y}) = \underset{k}{\operatorname{argmax}} \sum_{t=1}^{T} \alpha_t \cdot [\![\gamma_t(\mathbf{y}) = k]\!]. \tag{6.11}$$

6.3.2 Leave-one-out training

In the machine learning literature, it is common to split the dataset into training and test parts. Differing from those model-based classifiers, LRC requires a prototype set for outputting a prediction. An intuitive solution is to further split the training set into gallery and a smaller training set. However, the size of the gallery is critical to LRC; thus the further-split is usually unacceptable, especially when insufficient training samples are involved. We employ the leave-one-out strategy to utilize as many training instances as possible to construct the prototype set. In the training phase, each training data is classified, by LRC, based on the prototype set comprising all the other training samples. Then the training error is calculated according to the predicted labels. Consequently, the size of the prototype set in the training stage is always $N - 1$.

6.3.3 Random face patches

Generation of the face sub-patches is arbitrary in this work. We can exhaustively search the face image at all possible locations and scales, as in training a face detector [90]. Our preliminary experiments show that it is not necessary to use all the possibilities. Clearly fewer of candidates makes the training procedure faster. We generate the candidate patches as follows. Given a random position and width w. The height h of the patch is defined by $h = \sigma/w$, where σ is a predefined value of the patch area. In other words, we have required every patch to be the same size so that no single patch can

completely cover other ones. As a result, the weak classifiers based on these patches are less correlated, which is preferred by ensemble algorithms [5, 73]. Another reason for fixing the patch area is to control the complexity of the LRC classifiers. In the context of boosting algorithms, weak classifiers with low complexity may imply a higher generalization capability [24].

In this work, we empirically set $\sigma = 225$, $w \in \{5, 9, 15, 25, 45\}$ and $n_p = 2000$. This setting is already sufficient to obtain considerable improvement on recognition performance (see Section 6.6).

Random patches or random subspace? BLRC-P is highly robust to continuous occlusion. This is because the small patches are likely to be separated from the occlusion. Given that the area of occlusion is σ_o while that of the face is σ_f, it is trivial to see the probability that a patch overlaps the occlusion:

$$\Pr_{overlap} \approx \sigma_o / \sigma_f, \tag{6.12}$$

in the scenario that $\sigma_f > \sigma_o \gg \sigma$. In other words, a considerable portion of the patches are not influenced by the occlusion. In contrast, the random subspace is not so resistant to occlusions. Following the above setting, we can also easily get the probability that one random subspace (based on raw-pixels) [38] is influenced by occlusion is

$$\Pr_{overlap} = 1 - (1 - \sigma_o / \sigma_f)^{\sigma} \approx 1. \tag{6.13}$$

Furthermore, all the intact patches, and the corresponding LRC classifiers will be aggregated by the SAMME algorithm. The boosting strategy makes BLRC-P even more efficient and robust than the majority voting based methods in [98] and [57]. This analysis is also empirically proved by our experiments.

6.3.4 Accelerating the BLRC algorithm

It is easy to see the computation complexity of each patch-based LRC is

$$C_L = \mathcal{O}(N_k^3) + \mathcal{O}(N_k^2 \sigma), \tag{6.14}$$

then the complexity of the training procedure is given by

$$C_{train} = n_p \cdot T \cdot C_L = \mathcal{O}(n_p T N_k^3) + \mathcal{O}(n_p T N_k^2 \sigma), \tag{6.15}$$

where n_p is the number of patch candidates, T is the number of the selected ones. The computational burden of the prediction process is

$$C_{test} = T \cdot C_L = \mathcal{O}(T N_k^3) + \mathcal{O}(T N_k^2 \sigma). \tag{6.16}$$

The extremely high complexities prevent BLRC from being practical. Fortunately, with two novel modifications, we can reduce the complexity *significantly*. The fast version of BLRC is not only applicable to real-time applications, it also outperforms many state-of-the-art face recognizers in terms of efficiency.

Fast training For the conventional weak classifiers used in boosting, such as decision trees, decision stumps and the linear LDA classifier, one needs to re-train the weak classifier after the training samples' weights $\boldsymbol{\omega}$ are updated. We show that *this is not the case for LRC*.

Theorem 6.1. *The training procedure of LRC is independent of the training example's weights $\boldsymbol{\omega}$, given that $\boldsymbol{\omega} > 0$ (which is automatically satisfied in a boosting algorithm). In other words, in BLRC, all the weak classifiers need to be trained only once.*

Proof: let $\boldsymbol{\Omega}_k \in \mathbb{R}^{N_k \times N_k}$ be the diagonal matrix such that $\boldsymbol{\Omega}_k(i,i) = \boldsymbol{\omega}_k(i),\ i = 1, 2, \ldots, N_k$, where $\boldsymbol{\omega}_k$ is the weight vector for the training face images that belong to the kth class. By taking data weight into consideration, (6.1) is rewritten as

$$\min_{\hat{\boldsymbol{\beta}}_k} \|\mathbf{y} - \mathbf{X}_k \boldsymbol{\Omega}_k \hat{\boldsymbol{\beta}}_k\|^2, \forall\, k = 1, 2, \ldots, K. \tag{6.17}$$

The above quadratic problem comes with a closed-form solution that writes

$$\hat{\boldsymbol{\beta}}_k^{\star} = (\boldsymbol{\Omega}_k \mathbf{X}_k^{\top} \mathbf{X}_k \boldsymbol{\Omega}_k)^{-1} \boldsymbol{\Omega}_k \mathbf{X}_k^{\top} \mathbf{y} \tag{6.18}$$

and we know that

$$\boldsymbol{\omega} > 0 \implies \boldsymbol{\omega}_k > 0 \implies \boldsymbol{\Omega}_k^{-1} \text{ exists.} \tag{6.19}$$

(6.18) can be further rewritten into

$$\begin{aligned} \hat{\boldsymbol{\beta}}_k^{\star} &= \boldsymbol{\Omega}_k^{-1} (\mathbf{X}_k^{\top} \mathbf{X}_k)^{-1} \boldsymbol{\Omega}_k^{-1} \boldsymbol{\Omega}_k \mathbf{X}_k^{\top} \mathbf{y} \\ &= \boldsymbol{\Omega}_k^{-1} (\mathbf{X}_k^{\top} \mathbf{X}_k)^{-1} \mathbf{X}_k^{\top} \mathbf{y} \\ &= \boldsymbol{\Omega}_k^{-1} \boldsymbol{\beta}_k^{\star}, \end{aligned} \tag{6.20}$$

where $\boldsymbol{\beta}_k^{\star}$ is the solution to the unweighted LRC. We now can obtain the reconstruction residual $\hat{r}_k$ for class k as

$$\begin{aligned} \hat{r}_k &= \|\mathbf{X}_k \boldsymbol{\Omega}_k \hat{\boldsymbol{\beta}}_k^{\star} - \mathbf{y}\|^2 \\ &= \|\mathbf{X}_k \boldsymbol{\Omega}_k \boldsymbol{\Omega}_k^{-1} \boldsymbol{\beta}_k^{\star} - \mathbf{y}\|^2 = r_k. \end{aligned} \tag{6.21}$$

This result, without loss of generality, is valid for all the classes. Finally, we arrive at the prediction $\hat{l}^{\star}(\mathbf{y})$

$$\hat{l}^{\star}(\mathbf{y}) = \underset{k}{\operatorname{argmin}} \{\hat{r}_k\} = \underset{k}{\operatorname{argmin}} \{r_k\} = l^{\star}(\mathbf{y}). \tag{6.22}$$

That is to say, training data weights do not have any impact on the prediction of the LRC classifier.

Actually, a more intuitive understanding of the above analysis is in (6.17): if we see $(\Omega_k \hat{\boldsymbol{\beta}}_k)$ as the variable of interest, we solve exactly the same problem as the standard least squares fitting problem. □

Accordingly, one needs to calculate the prediction of the patch-based LRC classifiers only once. The predictions for the jth weak classifier are recorded as $\boldsymbol{\gamma}_j = [\gamma_j(\mathbf{x}_1), \gamma_j(\mathbf{x}_2), \cdots, \gamma_j(\mathbf{x}_N)]$, and the weighted error for this LRC predictor is given by $\sum_{i=1}^{N} \omega_i \cdot [\![\boldsymbol{\gamma}_j(i) \neq l_i]\!]$.

With the oracle $\boldsymbol{\gamma}_j, \forall j = 1, 2, \cdots, n_p$, the training cost is reduced by T times to

$$\tilde{C}_{train} = n_p \cdot C_L = \mathcal{O}(n_p N_k^3) + \mathcal{O}(n_p N_k^2 \sigma). \tag{6.23}$$

Usually, T is of order 10^2, so the above strategy can gain a speedup of a few hundred times. This desirable property makes the proposed BLRC very compelling in terms of computation efficiency.

Fast prediction LRC is already very efficient compared with many other FR methods. However, in BLRC, we need to perform a few hundred LRC classifiers for a strong predictor. We want to speed up the prediction procedure as well.

The intermediate matrices $\mathbf{A}_k = (\mathbf{X}_k(\mathbf{X}_k^\top \mathbf{X}_k)^{-1}\mathbf{X}_k^\top - \mathbf{I}), \forall\, k = 1, 2, \cdots, K$, are obtained off-line for all the patches. Then in the prediction, one calculates the reconstruction residual as

$$r_k = \|\mathbf{A}_k \mathbf{y}\|^2. \tag{6.24}$$

Thus, the computational burden of BLRC's prediction remarkably decreases to

$$\tilde{C}_{test} = \mathcal{O}(T N_k \sigma). \tag{6.25}$$

This is also an impressive result, which makes the BLRC algorithm easily applicable to real-time applications.

Finally, we conclude the BLRC-P in Algorithm 8.

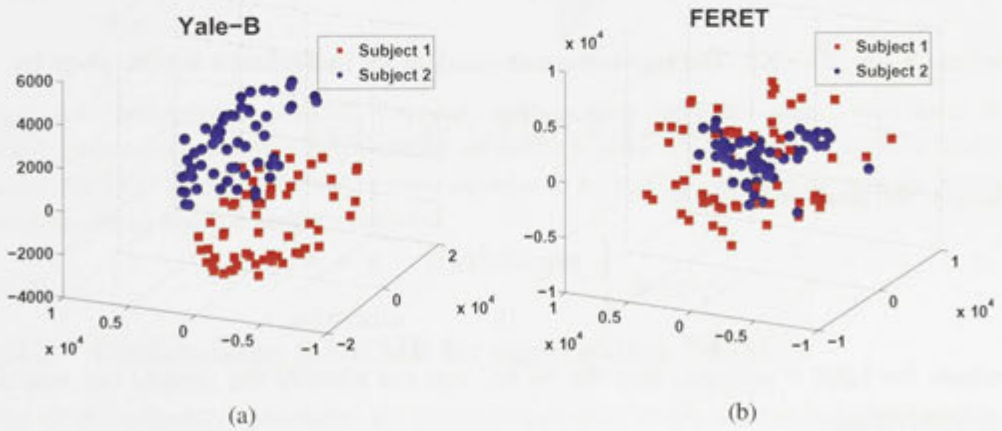
6.4 LRC Constrained by Local Neighborhoods

It is well known that facial images, belonging to the same subject, form a *non-linear* manifold with intrinsically low dimensionality in the feature space. The manifold, which accommodates different poses, illuminations and expressions, is not necessarily linear. The manifolds belonging to the subjects in Yale-B and FERET are illustrated

Algorithm 8 BLRC-Patch

1. Normalize the training faces $\mathbf{X}$ and test face $\mathbf{y}$.
2. Generate $\mathbf{n}_p$ face patches $\mathbb{P} = \{p_1, p_2, \cdots, p_{n_p}\}$ on the training images and the associated weak classifiers $\gamma_j \; \forall j \in \{1, 2, \cdots, n_p\}$.
3. Get all the predict labels $\gamma_j(\mathbf{x}_i) \; \forall i, j$.
4. Train the BLRC-P using SAMME, record the selected face patches and their weights $\alpha_t \; t = 1, 2, \cdots, T$.
5. Crop the selected patches for $\mathbf{y}$ and perform each patch-based weak classifier, get predictions $\gamma_j(\mathbf{y}) \; \forall j$.
6. Aggregate the predict labels using (6.11).

in Figure 6.2. When the faces are perfectly aligned (Yale-B), the linear subspace is sufficiently effective to distinguish them. However, when some disturbances are involved, the linear subspace is too primitive to accommodate the variations with respect to occlusion, pose and noise. In Figure 6.2(b) (FERET), it is obviously intractable to discriminate the two groups by simply using linear models.



(a) (b)

Figure 6.2: Face manifolds in the Principle-Component feature space. Note that the first PC is removed for roughly eliminating the global facial information.

6.4.1 Boosting LRC with neighborhoods

Local neighborhoods of faces In manifold learning, it is common to estimate the local region of a manifold using locally linear models, for example [69]. We therefore assume that the linear subspace model is only precise inside some certain locality. Under this assumption, we introduce the *local neighborhood of faces*. Given a face set $\mathbf{X} \in \mathbb{R}^{d\times N}$ and a randomly selected index v, a neighborhood of face v, denoted as $\mathcal{N}$, covers a local part of the data, *i.e.*,

$$\mathcal{N} \triangleq \{\forall\, \mathbf{z} \in \mathbb{R}^d \mid \|\mathbf{z} - \mathbf{x}_v\| < \delta_v^\kappa\}, \tag{6.26}$$

where δ_v^κ is the Euclidean distance between $\mathbf{x}_v$ and its κth nearest neighbor ($1 < \kappa < N$). For a reasonable κ, we can assume that the linear model is sufficiently accurate to describe the manifold structure in $\mathcal{N}$. To sufficiently cover all the possibilities for the given training database, a set $\mathbb{Q} = \{\mathcal{N}_1, \mathcal{N}_2, \cdots, \mathcal{N}_{n_l}\}$ consisting of n_l local neighborhoods is randomly generated. In practice, we simply set $n_l = 100$ and κ is randomly selected from $\{100, 150, 200, 250, 300\}$. The multiple scales of locality offer higher flexibility to represent the data distribution.

Neighborhood constrained LRC We now design a new LRC algorithm, termed *Neighborhood constrained LRC* (NLRC). Analogous to the patch-based LRC, the corresponding NLRC is determined by given a local neighborhood. In NLRC, only the faces inside a local neighborhood $\mathcal{N}$ are considered. Mathematically, the minimization problem in (6.1) is rewritten into:

$$\min_{\boldsymbol{\beta}_k} \|\mathbf{y} - \hat{\mathbf{X}}_k\boldsymbol{\beta}_k\|^2, \tag{6.27}$$

where $\hat{\mathbf{X}}_k = \mathcal{N} \cap \mathbf{X}_k$. The reconstruction residual for individual k is now given by

$$\hat{r}_k = \left\|(\hat{\mathbf{X}}_k(\hat{\mathbf{X}}_k^\top\hat{\mathbf{X}}_k)^{-1}\hat{\mathbf{X}}_k^\top - \mathbf{I})\mathbf{y}\right\|^2. \tag{6.28}$$

Lastly, the predicted label is

$$\gamma(\mathbf{y}) \triangleq \begin{cases} \underset{k}{\operatorname{argmin}}\{\hat{r}_k\}, & \mathbf{y} \in \mathcal{N}, \\ 0, & \text{otherwise}, \end{cases} \tag{6.29}$$

where the label 0 indicates that the NLRC can not identify the sample out side the neighborhood.

Random neighborhoods or random sampling? Both random-neighborhood and random sampling [94, 48, 95] are methods to choose subsets from original samples. However, we argue that the random-neighborhood approach is more effective and accurate when linear-representation based classifiers are involved.

To explain the superiority, in Figure 6.3, we generate two synthetic manifolds (blue and red) in the 3D space. The facial vectors from two different subjects distribute within the associated manifolds. Suppose a probe image $\mathbf{y}$, which is shown as a yellow point, belongs to subject 2. With considerable probability, it could lie exactly on the plane (red triangle) spanned by the feature points a_1, a_2, a_3 from subject 1, and lie very close to the blue triangle, which is the nearest linear subspace spanned by the samples from subject 2. According to the criteria of LRC, the label of $\mathbf{y}$ will be mis-assigned to be 1, despite how unrelated a_1 and a_3 are to the test face. The ambiguity exists, not only in LRC but also in all the other linear-representation-based manners, such as NFL, NFS and SRC.

Obviously, the random sampling method won't help in this scenario, because it treats all the examples equally, despite any spacial information. In contrast, NLRC focuses only on the samples in the limited region (inside the green dashed round), thus correctly classifying $\mathbf{y}$, as illustrated in Figure 6.3

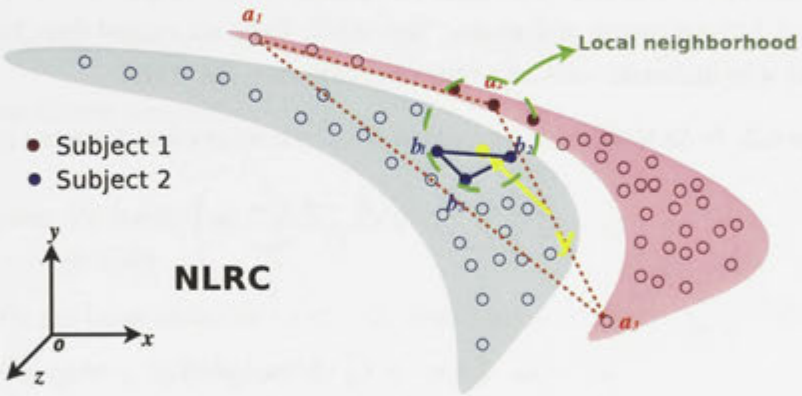


Figure 6.3: Demonstration of NLRC. The local neighborhood is shown as a green dashed round, the gallery images inside the local neighborhood are shown as solid, while others are shown as hollow circles. For LRC, the label of $\mathbf{y}$ will be mis-assigned as 1, while NLRC does not consider the faraway points $\mathbf{a}_1$ and $\mathbf{a}_3$, thus $\mathbf{y}$ is correctly classified.

6.4.2 Customizing SAMME for aggregating NLRC's

The NLRC algorithm identifies the probe image only inside a local neighborhood $\mathcal{N}$, *i.e.*

$$\|\mathbf{y} - \mathbf{x}_v\| < \delta_v^\kappa$$

Other neighborhoods that do not contain the probe image ideally should give the answer "I don't know." Because we have no prior knowledge about which neighborhood

should be used for fitting the linear regressor, we generate as many neighborhoods as possible and employ a boosting select and combine linear regressors defined on these generated neighborhoods. Mathematically, we want to aggregate those weak classifiers $\mathbb{H} = \{\gamma_1, \gamma_2, \cdots, \gamma_{n_l}\}$, which are generated from $\mathbb{Q}$. The new FR approach is termed BLRC-N and is illustrated in Figure 6.4.

Unfortunately, the standard SAMME algorithm cannot accommodate *uncertain weak classifiers* that output a decision "I don't know". We therefore need to derive a modified SAMME algorithm that can work with uncertain weak classifiers. The proposed new version of SAMME, called SAMME-U, is based on replacing the uncertain prediction by $\gamma(\mathbf{y}) = u$, which is the output of *random guessing*:

$$\Pr(u) = \begin{cases} (K-1)/K, & u \neq l_y; \\ 1/K, & u = l_y, \end{cases} \tag{6.30}$$

where l_y is the true label of $\mathbf{y}$. Schapire and Singer [73] extended standard binary-class AdaBoost to the case of uncertain weak classifiers, *i.e.*, the outputs of weak classifiers are $\{-1, 0, 1\}$. An output of 0 means "not sure". Here we extend their binary-class AdaBoost with uncertain weak classifiers to multi-class SAMME.

Theorem 6.2. *In SAMME-U, the weighted error of a weak classifier $\gamma(\mathbf{x})$ is given by*

$$\varepsilon = \frac{1}{Z}\left(\sum_{i=1}^{N} \omega_i^{-} + \frac{K-1}{K}\sum_{i=1}^{N} \omega_i^{0}\right), \tag{6.31}$$

where

$$\begin{aligned} \omega_i^{-} &= \omega_i \cdot [\![\gamma(\mathbf{x}_i) \neq l_i]\!] \cdot [\![\gamma(\mathbf{x}_i) \neq 0]\!], \\ \omega_i^{0} &= \omega_i \cdot [\![\gamma(\mathbf{x}_i) = 0]\!]; \end{aligned}$$

and $Z = \sum_{i=1}^{N} \omega_i$ is the normalization constant.

Proof: Considering the uncertain prediction $\gamma(\mathbf{x}_i) = \gamma_i^*$, the boolean function $[\![\gamma(\mathbf{x}_i) \neq l_i]\!]$ in Algorithm 6 is converted to the following expectation

$$\begin{aligned} \Psi(\mathbf{x}_i) &\triangleq \mathrm{E}([\![\gamma(\mathbf{x}_i) \neq l_i]\!]) \\ &= \Pr(\gamma(\mathbf{x}_i) = l_i) \cdot 0 + \Pr(\gamma(\mathbf{x}_i) \neq l_i) \cdot 1 \end{aligned} \tag{6.32}$$

Then we arrive at

$$\Psi(\mathbf{x}_i) = \begin{cases} (K-1)/K, & \gamma(\mathbf{x}_i) = 0; \\ 0, & \gamma(\mathbf{x}_i) = l_i; \\ 1, & \text{otherwise.} \end{cases} \tag{6.33}$$

By substituting (6.33) for the boolean function, we obtain (6.31) easily. □

As mentioned, the idea of utilizing uncertain predictions for boosting algorithm was firstly studied by Freund *et al.* [27, 73]. The SAMME-U method is summarized in Algorithm 9. Actually, we can prove the following proposition.

Proposition 6.4.1. *When SAMME-U is applied to binary-class problems ($K = 2$), SAMME-U is identical to AdaBoost with the uncertain weak classifiers of Schapire and Singer [73].*

Algorithm 9 SAMME-U: SAMME with uncertain predictors

Input:

- A training data-label set $(\mathbf{x}_1, l_1), \cdots, (\mathbf{x}_N, l_N)$.
- A data weight set $\omega_1, \cdots, \omega_N$.
- A functional space $\mathbb{H} = \{\gamma_j(\mathbf{x}), \gamma_j(\mathbf{x}), \cdots, \gamma_j(\mathbf{x})\}$.
- A maximum iteration T.

begin

· Initialize the weights $\omega_i = 1/N, \ \ i = 1, 2, \cdots, N$;

for $t \leftarrow 1$ **to** T **do**

· Fit the weak classifier $\gamma_t(\mathbf{x}) \in \mathbb{R}$, such that $\gamma_t = \underset{\gamma \in \mathbb{H}}{\operatorname{argmin}} \sum_{i=1}^{N} \omega_i \cdot \Psi(\mathbf{x}_i)$;

· Compute ε_t by using (6.31);

· $\alpha_t \leftarrow \log((1 - \varepsilon_t)/\varepsilon_t) + \log(K - 1)$;

· Set $\omega_i \leftarrow \omega_i \cdot \exp(\alpha_t \cdot \Psi(\mathbf{x}_i))$;

· Re-normalize ω_i;

· **if** converge, break;

end

Output: The strong classifier: $F(\mathbf{y}) = \underset{k}{\operatorname{argmax}} \sum_{t=1}^{T} \alpha_t \cdot [\![\gamma_t(\mathbf{y}) = k]\!]$.

In the stage of prediction, the candidate local neighborhoods are first collected and the results of the corresponding NLRCs are combined using (6.11), to identify the probe face. Note that in the context of BLRC-N, the strategies for faster training and test mentioned above are still applicable.

Lastly, the BLRC-N is summarized in Algorithm 10.

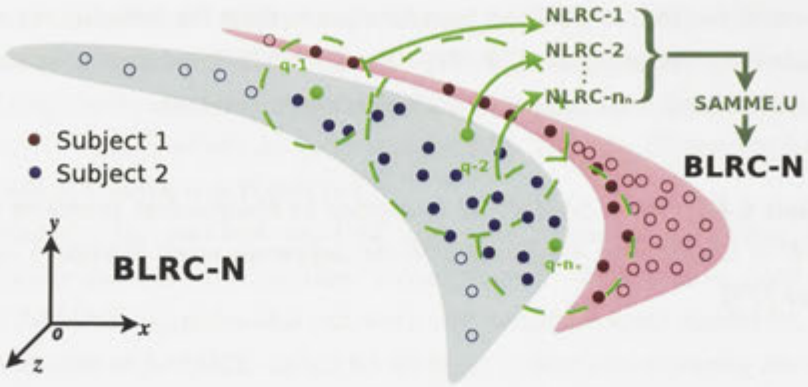


Figure 6.4: Demonstration of the BLRC-N algorithm. The randomly selected NLRC's are boosted by the customized SAMME algorithm. Note that the NLRC's are not necessarily identical in size and they sometimes overlap with each other.

6.5 Totally Corrective BLRC

6.5.1 BLRC with SAMME-CG

As described in the above chapters, totally corrective boosting usually brings higher performance and efficiency. In this section, we substitute the totally corrective SAMME for the original one to further improve the accuracy of face recognition. The BLRC with SAMME-CG here is denoted as BLRC-CG. We perform the BLRC-CG only with patch-based LRCs because the loss function of BLRC-N is changed and thus the SAMME-CG is no longer applicable for it.

The substitution of the training method is straightforward. Algorithm 7 is adopted instead of standard SAMME to select the LRCs and calculate their weight. Note that the leave-one-out strategy and the "one-off" training of LRCs remain unchanged. The experimental results show consistency with our previous conclusion that the CG-based algorithm usually outperforms its prototype due to the more flexible model.

6.5.2 BLRC with single-step

Both the standard AdaBoost and our CG variations are iteration based. The motivation of the iterative paradigm is to select weak classifiers among infinite or extremely large numbers of candidates. Nonetheless, only 2000 weak classifiers are involved in our algorithm and the number is empirically verified as sufficient. We then can discard the iterative framework and directly solve the primal 6.8 or dual problem 6.9 in a single-step. The new variation of BLRC, which is termed BLRC-batch, illustrates the highest

Algorithm 10 BLRC-Neighborhood

1. Normalize the training faces $\mathbf{X}$ and test face $\mathbf{y}$.
2. Generate $\mathbf{n}_p$ face neighborhoods $\mathbb{Q} = \{\mathcal{N}_1, \mathcal{N}_2, \cdots, \mathcal{N}_{n_l}\}$, approximately covering the distribution of $\mathbf{X}$
3. Generate the associated weak classifiers $\gamma_j \; \forall j \in \{1, 2, \cdots, n_l\}$.
4. Get all the predict labels $\gamma_j(\mathbf{x}_i) \; \forall i, j$.
5. Train the BLRC-N using SAMME-U, record the selected neighborhoods and their weights $\alpha_t \; t = 1, 2, \cdots, T$.
6. Select the gallery faces according to the chosen neighborhoods and perform each neighborhood-based weak classifier to get $\gamma_j(\mathbf{y}) \; \forall j$.
7. Aggregate the predict labels using (6.11).

accuracy in the experiment while slightly inferior efficiency is also observed.

In summary, the general procedure of BLRC-CG and BLRC-batch is shown in Algorithm 11.

6.5.3 Faster model selection

Another issue arising here is how to select a proper parameter θ among the candidates for both BLRC-CG and BLRC-batch. When a set of parameter candidates is available, the validation method such as cross-validation is usually used to select the optimal one. Generally speaking, n $(n > 1)$ rounds of validation procedures are conducted over different splits of training samples. This makes the training extremely slow when n is large.

Fortunately, the leave-one-out training of BLRC provides another merit: there is no need to tune the model-parameter via a cross-validation process. In the context of model-based learning, one uses the cross-validation error, rather than the training error, to be the selection criterion. This is because the model overfits the training set, thus the training error can not serve as a good estimate for the generalization error. In contrast, the cross-validation error could be viewed as the approximated leave-one-out error, which is a unbiased estimate for the generalization error [22]. However, this is not the case for our algorithm: here, the training error of BLRC is obtained via the leave-one-out procedure. Every prediction of a training sample is generated without its own

Algorithm 11 Totally corrective BLRC

1. Normalize the training faces $\mathbf{X}$ and test face $\mathbf{y}$.
2. Generate $\mathbf{n}_p$ face patches $\mathbb{P} = \{p_1, p_2, \cdots, p_{n_p}\}$ on the training images and the associated weak classifiers $\gamma_j \; \forall j \in \{1, 2, \cdots, n_p\}$.
3. Get all the predict labels $\gamma_j(\mathbf{x}_i) \; \forall i, j$.
4. Perform totally corrective SAMME with all the parameter candidates $\{\theta_1, \theta_2, \cdots, \theta_U\}$:
   for $u \leftarrow 1$ **to** U **do**
   · BLRC-CG: Train the ensemble model using SAMME-CG;
   · BLRC-batch: Train the ensemble model via solving 6.9;
   · Record the selected face patches and their weights $\boldsymbol{\alpha}^u$;
   · Calculate the training error $\varepsilon_u = \frac{1}{N} \sum_{i=1}^{N} [\![\gamma_t(\mathbf{x}_i) \neq l_i]\!]$;
   end
5. Select the model index $u^\star$ such that $u^\star = \underset{u}{\operatorname{argmin}} \, \boldsymbol{\varepsilon}$.
6. Retrieve the selected face patches and the weights associated to the selected model.
7. Crop the selected patches for $\mathbf{y}$ and perform each patch-based weak classifier, get predictions $\gamma_j(\mathbf{y}) \; \forall j$.
8. Aggregate the predict labels using (6.11).

information, *i.e.* every training sample is predicted by using its complementary set. In this scenario, the overall training error of the ensemble won't be biased too much. To some extent, we could consider the training error of BLRC as an approximation of the leave-one-out error, and select the optimal parameter depending on it.

From the perspective of the instance-based learning, cross-validation is also unacceptable. Unlike model-based learning algorithms, the instance-based learning is more sensitive to the size of instance set (*i.e.* the face gallery in this chapter). Galleries with different sizes might lead to totally different model parameters. However, the n-fold cross-validation requires to split the training set into different parts. That is to say, in every "fold", we only use one part of the gallery. The size difference between the whole gallery and the partial gallery is not trivial when $n \ll N$ (otherwise it becomes

a leave-one-out validation). Recalling that the final model is obtained based on all the training samples, it makes little sense to impose the "optimal" parameter, which is selected for partial galleries, on the entire gallery model.

The validity of the direct usage of the training error is proved empirically in the experimental part. We tune the parameter for both BLRC-CG and BLRC-batch, no significant overfitting is observed.

6.6 Experiments

6.6.1 Experiment setting

We design a series of experiments for evaluating both accuracy and efficiency of the proposed FR algorithms. We compare our algorithms with Nearest Feature Line (NFL) [44], Sparse Representation Classification (SRC) [98] and Linear Regression Classification (LRC) [57], which represent state-of-the-art approaches using linear models. Experiments are conducted on three datasets, namely, Yale-B [34], AR [51] and FERET [59]. Each dataset is *randomly* split into two parts: training and test sets. For each data split, random projection (randomfaces) [98], PCA (Eigenfaces) [87] and LDA (Fisherfaces) [46] are used to reduce the dimensions to 20, 40, 50, 100, 200 and 300. Every training and test faces are normalized so that $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$. On each dimension, the test is repeated 5 times and we report the average and standard deviation. All the FR methods are conducted in Matlab, on a PC with a 2.6GHz quad-core CPU (only one core is used) and 4GB RAM.

The parameters for Random face patches and random face neighborhoods are selected as described in Section 6.3.3 and Section 6.4.1 respectively. We select the trade-off parameter θ in BLRC-CG and BLRC-batch among the candidates $\{100, 200, 500, 700, 1000, 1500, 2000, 5000, 7000, 10000\}$. Besides the standard BLRC-P (with 225-D patches) implementation, we also perform dimension-reduction on the patches for efficiency. That is to say, we reduce the dimensionality of the patch, from original 225 to the lower ones. This variation of BLRC-P is referred to as "BLRC-PD" hereafter. The BLRC-CG and BLRC-batch are also performed on the dimensionally reduced space, as implied by the context. The BLRC-N method, on the other hand, is directly performed on the extracted features (50-D Eigenfaces). The conventional majority voting method based on random subspace and sub-sampling is also tested, to highlight the superiority of the proposed boosting framework. For a fair comparison, we also perform the modular-based DEF approach [57], which uses simple heuristics to partition a face and run LRC for each parts, on the AR database. The inferiority of

this primitive apprach is also reported, compared with the proposed methods.

6.6.2 Experiments on well-aligned faces with no occlusion

Yale-B is a dataset for FR which contains $2,414$ well-aligned face images from 38 individuals under various lighting conditions, as illustrated in Figure 6.5. For each subject, we randomly choose 30 images to compose the training set and other 30 images for testing. Note that the BLRC-PD is not conducted for 200-D and 300-D data considering that the dimensionality of facial patches (225) is smaller than 300 and too close to 200. The Fisherfaces are only generated with dimensionality 20 and 35 because of the requirement of LDA that reduced dimensionality should be always smaller than the class number. We deliberately perform LRC with 20-D features, which does not meet LRC's requirement, to justify the efficacy of the proposed boosting framework.

Figure 6.5: The demonstration of Yale-B dataset with extreme illumination conditions.

The experiment results are reported in Table 6.1. We report the results of 35-D Fisherfaces in the 40-D column. As can be seen, for Fisherfaces, SRC shows better accuracy over other methods with the best accuracy of 95.1%. Except for this, our algorithms (BLRC-PD, BLRC-P, BLRC-CG and BLRC-batch) consistently outperform other compared approaches. In particular, the BLRC-batch achieves an accuracy of 99.6% on the 225-D feature space. To our knowledge, *this is the highest recognition rate ever reported for Yale-B with a similar experiment setting*. Moreover, BLRC-CG and BLRC-batch both illustrate remarkable excellence on randomfaces, with the highest recognition rates of 99.3% and 99.5% respectively. The original BLRC-P algorithm performs slightly worse than its cousins but still achieves the accuracy of 98.8%. The voting strategy (with 2000 LRCs based on 225-D face patches) outperforms all state-of-the-art but is obviously inferior to our algorithms. In this sense, we may conclude

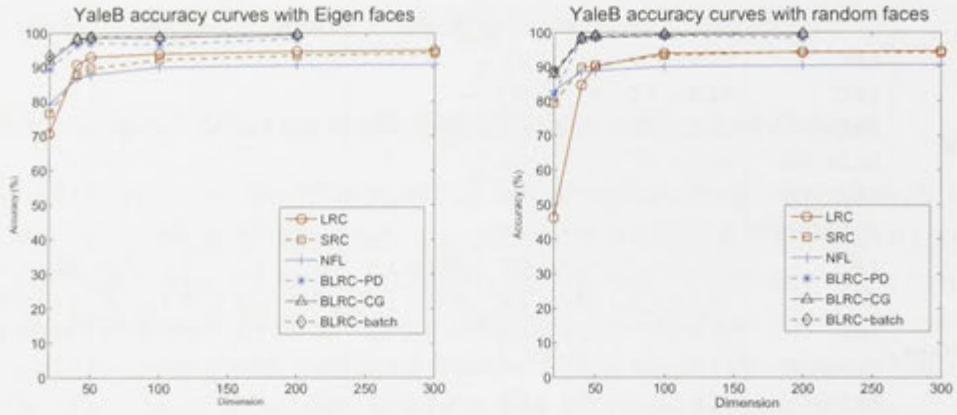
		D-20	D-40	D-50	D-100	D-200	D-300
LDA	LRC	$26.6 \pm 1.8^{\star}$	68.3 ± 2.4	-	-	-	-
	SRC	$\mathbf{92.2 \pm 1.7}$	$\mathbf{95.1 \pm 1.7}$	-	-	-	-
	NFL	89.3 ± 2.1	93.8 ± 1.8	-	-	-	-
	BLRC-PD	30.3 ± 4.7	85.7 ± 1.8	-	-	-	-
	BLRC-CG	38.1 ± 6.5	87.3 ± 2.9	-	-	-	-
	BLRC-batch	37.8 ± 7.5	92.9 ± 1.2	-	-	-	-
Random	LRC	$46.2 \pm 4.4^{\star}$	84.7 ± 2.2	90.1 ± 1.6	93.9 ± 1.2	94.1 ± 1.0	94.5 ± 1.3
	SRC	79.4 ± 2.2	89.7 ± 2.0	90.4 ± 2.0	93.4 ± 2.0	94.3 ± 1.8	94.0 ± 1.1
	NFL	83.0 ± 2.0	88.6 ± 1.8	88.8 ± 2.3	90.0 ± 2.0	90.3 ± 1.4	90.5 ± 1.6
	BLRC-PD	81.9 ± 3.3	98.2 ± 1.3	98.5 ± 1.0	98.8 ± 1.0	-	-
	BLRC-CG	87.9 ± 2.7	98.1 ± 0.8	98.8 ± 0.4	99.3 ± 0.3	-	-
	BLRC-batch	$\mathbf{88.7 \pm 2.4}$	$\mathbf{98.5 \pm 0.5}$	$\mathbf{99.1 \pm 0.3}$	$\mathbf{99.5 \pm 0.3}$	-	-
Eigen	LRC	$70.6 \pm 2.5^{\star}$	90.6 ± 1.4	92.8 ± 1.4	93.8 ± 1.3	94.5 ± 1.2	94.8 ± 1.2
	SRC	76.5 ± 2.4	87.6 ± 2.1	89.5 ± 1.9	92.1 ± 1.1	93.4 ± 1.0	94.1 ± 1.1
	NFL	79.2 ± 1.6	86.8 ± 2.4	87.7 ± 2.5	89.9 ± 1.9	90.5 ± 1.8	90.7 ± 1.7
	BLRC-PD	89.4 ± 3.2	96.4 ± 1.5	97.0 ± 1.4	96.6 ± 1.5	-	-
	BLRC-CG	91.9 ± 2.9	98.0 ± 0.7	98.6 ± 0.3	98.7 ± 0.4	-	-
	BLRC-batch	$\mathbf{93.2 \pm 1.0}$	$\mathbf{98.3 \pm 0.3}$	$\mathbf{98.8 \pm 0.2}$	$\mathbf{98.8 \pm 0.3}$	-	-
BLRC-P		98.3 ± 1.6					
BLRC-CG		99.5 ± 0.6					
BLRC-batch		$\mathbf{99.6 \pm 0.3}$					
random+voting		95.7 ± 1.2					

Table 6.1: The comparison of accuracy on YaleB. The best performances are shown in bold. "Random+Voting" means LRC on randomly selecting pixels with the final result being reported by majority voting. BLRC-CG and BLRC-batch are performed with reduced dimensionality as well as with the original one (225-D). The $\star$ symbol indicates that the dimensionality is too low to meet LRC's requirement, and thus poor performances are usually observed.

that the proposed algorithms are likely to be superior to other linear methods on this extreme illumination dataset. Figure 6.6 shows the accuracy rates for randomfaces and Eigenfaces, as the dimensionality increases. The excellence of the proposed algorithms could be observed more easily: the curves of our algorithms are remotely above other ones.

Figure 6.7 shows the boosting procedure, *i.e.* the training and test error curves of BLRC-PD, BLRC-CG and BLRC-batch[‡]. Note that BLRC-batch is a single-step method; thus the corresponding curves are flat. For training and test errors of the other two algorithms, we observe a fast decrease, which indicates the efficacy of the proposed boosting patch approach. Specifically, BLRC-PD illustrates slower training speed than BLRC-CG while stops earlier (at the 42th iteration) as the training error be-

[‡]Here the three algorithms are all performed on the 50-D feature space which is generated by PCA



(a) (b)

Figure 6.6: Accuracy curves of Yale-B with Eigen (a) and random (b) faces, as dimensionality increases.

comes small enough. In contrast, BLRC-CG converges faster but keeps optimizing the loss function after the training error approaches 0. Higher empirical error is observed with BLRC-CG when convergence is achieved. BLRC-batch shows the best training and test performances which could be viewed as lower boundaries of the errors for the other two methods.

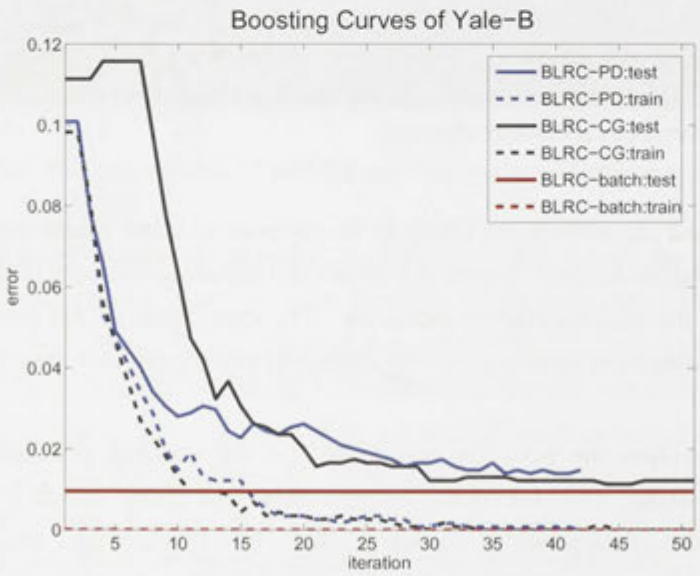
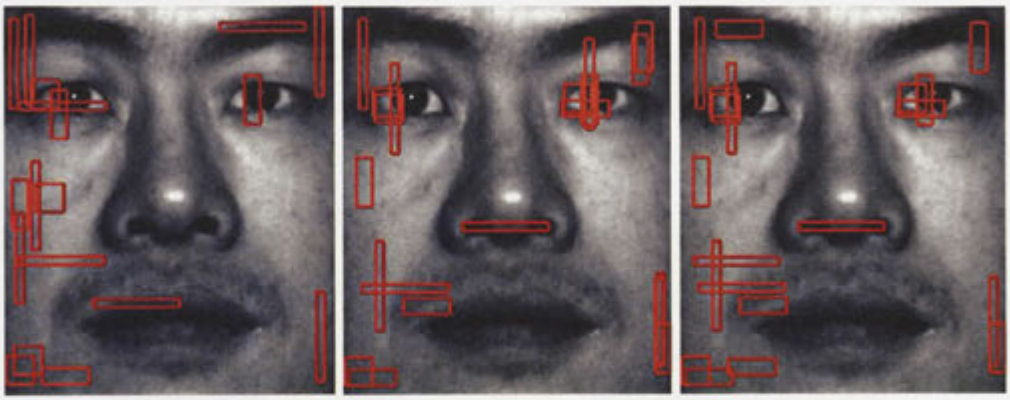


Figure 6.7: Demonstration of the boosting procedure of BLRC-PD, BLRC-CG and BLRC-batch with 50-D randomfaces (Yale-B). Note that BLRC-batch is a single-step method and BLRC-PD stop the training at iteration 42.

To illustrate how BLRC works, Figure 6.8 shows the most important 20 patches selected by BLRC-PD (Figure 6.8(a)), BLRC-CG (Figure 6.8(b)) and BLRC-batch (Figure 6.8(c)). We can see that the three algorithms select similar patches, with minor differences. The BLRC-PD pays more attention to the cheek while BLRC-CG and BLRC-batch focus on the eyes. The different choices regarding to patches lead to the performance gap.



(a) (b) (c)

Figure 6.8: The patches (shown as red blocks) selected by BLRC-PD (a), BLRC-CG (b) and BLRC-batch (c). The three images are generated by using the same face and patches selected by various algorithms on 50-D randomfaces.

Given $1,140$ faces are involved in the experiment and the recognition rate for BLRC-batch is 99.6%, only 5 faces are incorrectly classified in average. We select one BLRC-batch model with the error rate of 99.56%, that is the closest one to the mean (99.6%). The 5 misidentified faces are shown in Figure 6.9. *The faces are all unrecognizable for ourselves. In this sense, our algorithm actually perfectly solves the FR problem where faces are well-aligned.*

(a) (b) (c) (d) (e)

Figure 6.9: The 5 incorrectly classified faces by BLRC-batch. None of them are recognizable as human beings.

6.6.3 Experiments on faces with occlusion

There are 100 individuals in the AR (cropped version) dataset. Each subject consists of 26 face images which come with different expressions and considerable occlusions such as scarf and sunglasses (see Figure 6.10). In our setting, 13 images are randomly selected for training while the remaining ones are test images. We also test the modular-based DEF approach [57], which uses simple heuristics to partition a face and run LRC for each parts. In our experiment, we employ the 6-partition paradigm which achieves the best performance in their setting. The experiment results are reported in Table 6.2. Besides, the method employ random patches and voting strategy is performed.

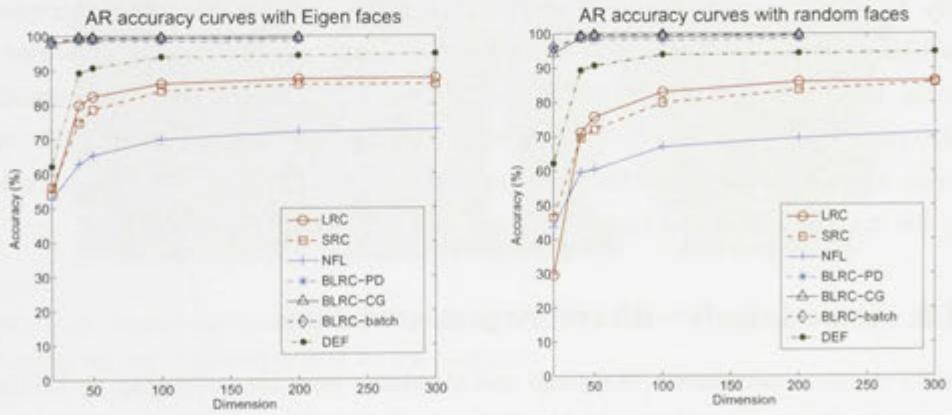
Figure 6.10: Images with occlusion in AR dataset. Note that we use only gray-scale faces in the experiment.

Similar to the experiment on Yale-B, the proposed boosting framework still overfits Fisherfaces. Except for this case, our methods show overwhelming superiority. *The BLRC-batch algorithm achieves a recognition rate of* 99.6% *which is also the best reported result on AR*. The performance gap between our methods and other competitors is commonly larger than 15% and peaks at 66.5% (LRC *vs*. BLRC-PD for 20-D randomfaces). The random subspace method (with voting) and DEF also show their robustness to the occlusion, while still inferior to the proposed methods. The more sophisticated approaches could select and combine the weak classifiers more efficiently, compared with the primitive ones, *i.e*. majority voting. Thus, the conclusion is that *the Boosted LRC can handle occlusion much better than other existing methods*. Figure 6.11 shows the accuracy rates as the dimensionality increases.

Figure 6.7 shows the boosting procedures for BLRC-PD, BLRC-CG and BLRC-

		D-20	D-40	D-50	D-100	D-200	D-300
LDA	LRC	49.9 ± 6.4	87.5 ± 6.6	91.4 ± 5.9	93.1 ± 3.9	-	-
	SRC	$\mathbf{87.0 \pm 5.3}$	93.2 ± 3.8	94.4 ± 3.6	95.0 ± 3.0	-	-
	NFL	83.9 ± 7.3	$\mathbf{93.3 \pm 5.1}$	$\mathbf{94.9 \pm 4.3}$	$\mathbf{96.0 \pm 3.7}$	-	-
	BLRC-PD	57.5 ± 3.9	72.6 ± 8.4	72.1 ± 6.0	73.4 ± 6.4	-	-
	BLRC-CG	58.3 ± 1.8	84.5 ± 2.0	87.7 ± 2.2	89.6 ± 1.9	-	-
	BLRC-batch	65.3 ± 2.2	85.4 ± 1.4	88.8 ± 3.8	89.3 ± 2.0	-	-
Random	LRC	29.7 ± 2.6	71.4 ± 7.5	75.9 ± 7.3	83.2 ± 7.2	86.2 ± 7.0	86.5 ± 6.8
	SRC	46.6 ± 3.8	69.8 ± 5.2	72.3 ± 5.1	80.1 ± 5.8	83.8 ± 5.7	86.0 ± 7.1
	NFL	43.8 ± 4.3	59.8 ± 6.1	60.6 ± 7.4	67.2 ± 8.4	69.8 ± 8.7	71.4 ± 7.4
	BLRC-PD	$\mathbf{96.2 \pm 2.4}$	98.6 ± 1.1	98.4 ± 1.2	98.3 ± 1.6	-	-
	BLRC-CG	94.2 ± 1.7	98.7 ± 0.7	99.0 ± 0.3	99.1 ± 0.3	-	-
	BLRC-batch	95.9 ± 1.7	$\mathbf{99.2 \pm 0.3}$	$\mathbf{99.6 \pm 0.3}$	$\mathbf{99.6 \pm 0.1}$	-	-
Eigen	LRC	54.2 ± 5.0	80.1 ± 6.3	82.6 ± 6.4	86.3 ± 6.8	87.6 ± 6.6	87.9 ± 6.8
	SRC	56.3 ± 4.5	75.0 ± 3.8	78.8 ± 4.0	84.2 ± 6.4	86.1 ± 7.3	86.1 ± 7.2
	NFL	52.9 ± 7.2	63.2 ± 8.5	65.6 ± 8.3	70.2 ± 8.3	72.3 ± 8.4	72.9 ± 8.5
	BLRC-PD	98.4 ± 1.0	98.5 ± 0.9	98.6 ± 1.4	98.5 ± 1.3	-	-
	BLRC-CG	97.7 ± 1.3	99.2 ± 0.7	99.1 ± 0.6	99.2 ± 0.4	-	-
	BLRC-batch	$\mathbf{98.4 \pm 1.0}$	$\mathbf{99.4 \pm 0.4}$	$\mathbf{99.4 \pm 0.5}$	$\mathbf{99.4 \pm 0.3}$	-	-
DEF		62.3 ± 5.3	89.4 ± 3.1	90.8 ± 2.3	94.0 ± 0.9	94.4 ± 1.6	94.8 ± 1.4
BLRC-P		98.7 ± 1.2					
BLRC-CG		99.3 ± 0.4					
BLRC-batch		$\mathbf{99.5 \pm 0.2}$					
random+voting		90.0 ± 6.2					

Table 6.2: The comparison of accuracy on AR dataset. In DEF, facial blocks are reduced to the dimension using down-sampling method. The results with 100-D fisherfaces are actually estimated with 95-D features because the class number is 100.



(a) (b)

Figure 6.11: Accuracy curves of AR with Eigen and random faces.

batch. Here BLRC-batch still shows as horizontal lines and all the error curves converge to a similar limit which reflects essentially the same loss function.

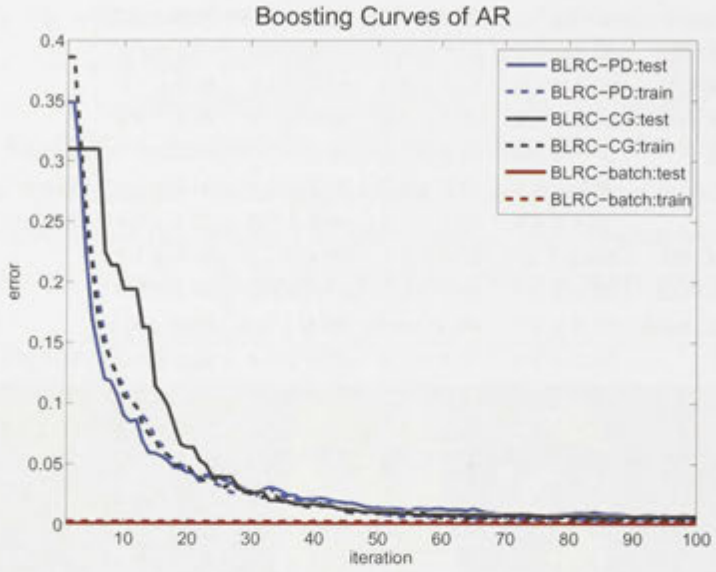


Figure 6.12: Demonstration of the boosting procedure of BLRC-PD, BLRC-CG and BLRC-batch with 50-D randomfaces (AR). Note that BLRC-batch is a single-step method and BLRC-PD stop the training at iteration 42.

Figure 6.13 shows the most important 20 patches selected by BLRC-PD (Figure 6.13(a)), BLRC-CG (Figure 6.13(b)) and BLRC-batch (Figure 6.13(c)) respectively. Compared with Figure 6.8, the most significant difference is that the proposed algorithms rarely choose the patches around the mouth, which sometimes covered by scarves. In other words, the patches are discarded as the occlusion pattern is learned. It is also interesting to see that from left to right, less and less attention is paid to the eyes region, where continuous occlusion (sunglasses) usually locates. This may explain why the accuracies increase one after one in the same order.

6.6.4 Experiments with severe pose variations

FERET dataset comprises $7,800$ head-and-shoulders portraits belonging to 739 subjects, as demonstrated in Figure 6.14. The portrait faces present significant variations in posture. We perform the BLRC-N algorithm on FERET. To assure sufficient training faces, we collect all the 20 categories with more than 33 images. 20 images (each person) are randomly selected for training and 13 portraits for test. *We stress that we do not perform any pre-processing (manual cropping or aligning) for the faces in FERET*

(a) (b) (c)

Figure 6.13: The random patches (shown as red blocks) selected by BLRC-PD, BLRC-CG and BLRC-batch. Most selected patches do not overlap with the occlusions (here sunglasses and scarves). The strong classifiers are trained on 50-D using random projection.

in this experiment. The raw images make the recognition task much more difficult than experiments on manually cropped and aligned faces using the same dataset, such as [48] and [93]. The results are reported in Table 6.3. Note that BLRC-N is performed with 50-D Eigenfaces and a sub-sampling method.

Figure 6.14: Images with severe pose variations in FERET dataset. Note that portraits for the same person are not necessarily captured in the same scenario.

According to the table, our algorithm achieves the highest accuracy (81.2%) again. This result supports our emphasis on the locality. Figure 6.15 (bottom) demonstrates that the training and test errors decrease quickly in iterations. In particular, the performance of the weak classifier (NLRC) is remarkably boosted, with the test error dropping from 80% to 20%.

		D-20	D-40	D-50	D-100	D-200	D-300
Random	LRC	$16.5 \pm 0.0^{\star}$	62.0 ± 2.3	68.7 ± 4.0	74.3 ± 2.0	76.0 ± 2.6	77.5 ± 2.8
	SRC	66.2 ± 2.7	67.5 ± 3.1	70.6 ± 2.6	74.0 ± 1.6	73.0 ± 2.0	72.5 ± 2.8
	NFL	67.7 ± 3.2	70.1 ± 4.3	72.1 ± 3.4	75.8 ± 2.3	76.4 ± 3.6	77.7 ± 2.2
Eigen	LRC	$10.7 \pm 2.1^{\star}$	79.6 ± 3.1	79.4 ± 3.5	79.7 ± 1.9	79.3 ± 2.3	78.2 ± 2.6
	SRC	70.2 ± 2.8	78.2 ± 2.1	79.2 ± 2.8	78.7 ± 1.6	75.9 ± 2.1	74.0 ± 2.1
	NFL	69.7 ± 3.0	78.2 ± 2.8	78.5 ± 3.0	78.2 ± 3.0	78.3 ± 2.7	78.5 ± 2.6
BLRC-N		$\mathbf{81.2 \pm 1.5}$					
Sampling		79.8 ± 3.0					

Table 6.3: Comparison of recognition results on FERET dataset. Sampling means randomly selecting a subset from the entire training samples to build LRC classifiers and vote for the final result. We don't perform the test with 20-D face features because LRC requires the dimension of the feature to be larger than the number of training images belonging to one individual. The $\star$ symbol indicates that the dimensionality is too low to meet LRC's requirement, thus poor performances are usually observed.

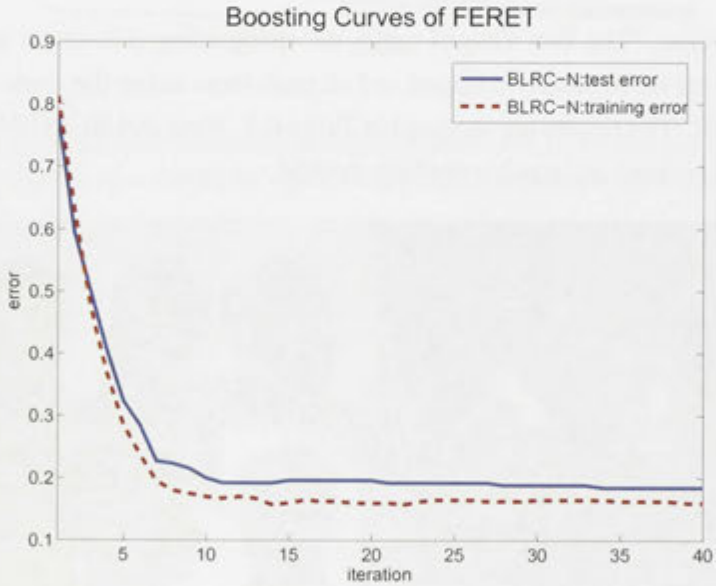


Figure 6.15: Demonstration of the boosting procedure of BLRC-P with 50-D randomfaces (FERET).

6.6.5 Efficiency comparison

For a computer vision algorithm, the running speed is usually critical. Here we show the efficiency of all the compared FR methods. Figure 6.16 depicts the running time (ms) of the compared methods. Most compared algorithms are performed on random-

faces with different dimensions for 100 times[§] and the average values are reported. Note that BLRC-N is always with 50-D Eigenfaces. To evaluate the fast prediction strategy, which is described in Section 6.3.4, BLRC-PD, BLRC-CG, BLRC-batch, BLRC-N and LRC are all re-tested in the faster manner. The faster methods are denoted by starting the original names with the "F-" prefix.

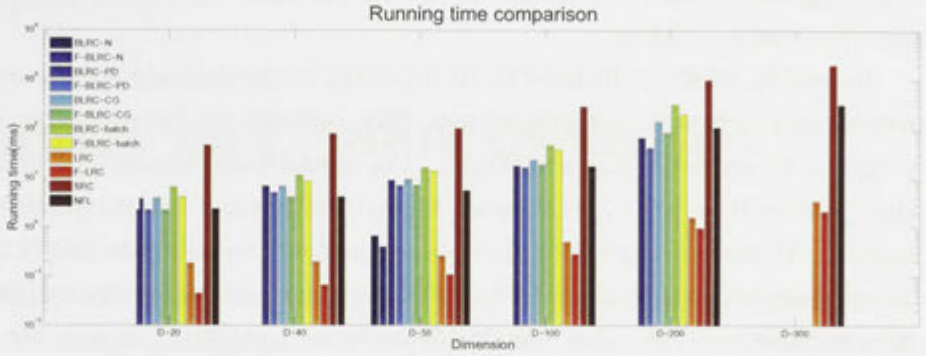


Figure 6.16: Comparison of running time. Methods starting with "F-" indicates that the algorithms are performed with the fast prediction strategy. Note that the y-axis is in log-scale.

As demonstrated, SRC is the slowest one, with processing time up to around $2,000$ ms per face due to the need to solve a convex optimization problem. The NFL also shows low efficiency (processing time above $1,000$ ms). In contrast, LRC shows extremely high efficiency, especially with the acceleration technique. The F-LRC, *i.e.*, the LRC algorithm with fast prediction, achieves a speed less than 0.1 ms per image. Meanwhile, our methods all illustrate the capacity for real-time applications. In particular, one only needs 0.7 ms to classify a face using the F-BLRC-N algorithm. That is because for each test face $\mathbf{y}$, only the selected NLRC's are involved in computation. In addition, most of the patch-based BLRC algorithms could be performed under 100 ms, which could be viewed as the real-time criterion. Please note that this experiment is conducted in Matlab and only one core of the CPU is used.

6.7 Conclusion

Face recognition methods based on linear representations have been shown to deliver state-of-the-art performance. However, two fundamental problems have not been addressed properly in the literature. The first problem is, which parts of the face images

[§]The patch-based algorithms are not performed on 200-D randomfaces; we use the original dimensionality (225) instead.

are more reliable for performing the linear representations; the other one is, which training faces could be selected as the representation basis.

In this work, we have proposed several boosting-like face recognition algorithms to address the above problems. The proposed methods can alleviate the difficulties of noise, occlusion and severe poses in face recognition. The methods are built upon the linear regression based face recognizer: we use the linear regression classifier as weak classifiers and boost them.

In specific, BLRC-P, BLRC-CG, BLRC-batch are the three algorithms to select the reliable face parts w.r.t. linear regressions. They optimize the same loss function while employ different optimization strategies. The experimental results on YaleB and AR show that totally corrective framework is empirically better than the gradient-descent method. All the best experimental results are achieved by using the totally corrective boosting approaches. The BLRC-N, on the other hand, determines the weights of random neighborhoods by using a modified version of SAMME. This way, the manifold of face image distribution is somehow learned in a boosting-like fashion. In other words, it selects the important bases of linear representations for the voting procedure. The experiment on the challenging datasets FERET shows the superiority of BLRC-N.

Chapter 7

Summary and Future Directions

In this thesis, we study the boosting algorithm from the perspective of convex optimization and margin distribution. We empirically show that margin distribution is significant to the explanation of AdaBoost's success. To further explain the importance theoretically, the Lagrange dual problems of AdaBoost, LogitBoost and soft-margin LPBoost with generalized hinge loss are derived. We show that they are all entropy regularized LPBoost, and thus the success of AdaBoost relies on maintaining a better margin distribution.

Based on the dual formulation, a general column generation based optimization framework is proposed. This optimization framework can be applied to solve the boosting algorithms with various loss functions. Consequently, we propose a series of boost-like algorithms based on column generation. AdaBoost-CG solves the dual problem of the original loss function of AdaBoost; MDBoost directly optimizes the margin distribution by maximizing the average margin and at the same time minimizing the margin variance; the AnyBoost$_{\mathrm{TC}}$, on the other hand, is an abstract totally corrective boosting framework that can be used to minimize a broad range of regularized risk. An objective in the form of an arbitrary convex loss function plus an arbitrary convex regularization term can be minimized using the boosting technique developed here. The superiorities of the proposed methods, in terms of efficiency or performance, are all justified via theoretical proof or empirical test. In specific, the totally corrective optimization strategy demonstrates higher convergence rate than the coordinate descent method used in AdaBoost. AdaBoost-CG achieves comparable test accuracies than AdaBoost as they essentially optimize the similar loss functions. As a empirical evidence that margin distribution is more important than the minimum margin, AdaBoost-QP, which directly optimizes the margin distribution (in terms of average margin and margin variance), outperforms AdaBoost and LPBoost in most cases while

usually generates smaller minimum margins. MDBoost, which is totally corrective and also directly optimizes the margin distribution, shows higher performances in terms of both convergence speed and accuracy than original AdaBoost.

Finally, we evaluate the totally corrective boosting algorithms in a practical computer vision application: face recognition. Four ensemble algorithms for face recognition, namely BLRC-P, BLRC-CG, BLRC-batch and BLRC-N are proposed. In a nutshell, we use the LRCs as weak classifiers and boost them. Benefiting from the sophisticated ensemble learning method, *i.e.* the totally corrective boosting algorithm, some record-breaking accuracies on famous datasets (YaleB, AR, FERET) are reported. Furthermore, we show that the totally corrective manner is empirically better than the traditional method.

We summarize the connections between our main contributions in Figure 7.1 where the contributions are emphasized as shaded rectangles.

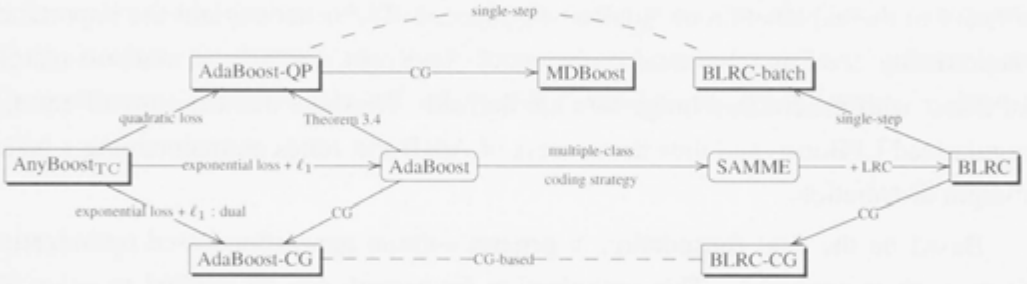


Figure 7.1: The diagram summarizes the connections between the proposed algorithms. In the diagram, "CG" denotes the column-generation method while ":dual" stands for solving the problem in the dual formulation. Note that the arrows may only indicate similarities between two methods.

Besides the algorithms we present in this thesis, some extended works have also been proposed: inspired by MDBoost, we (Shen *et al.* [82]) converted the LDA criterion into a boost-like problem and perform it for face detection, we achieved better performance than the state-of-the art methods; Hao *et al.* [37] extended the totally corrective framework for multiple-class problems, and the novel algorithms also outperformed the ordinary boosting approaches.

Several future topics are interesting and promising. First of all, the selection of the trade-off parameter of AdaBoost-CG influences the final performance significantly, although how to tune the parameter efficiently remains an unsolved problem.

For MDBoost, a future research direction is how to integrate useful prior information into the matrix A. We believe that improved performance may be obtained by carefully designing A. For example, one can take asymmetric data distribution

into consideration by giving a weight to each margin ρ_i, $i = 1 \cdots M$. Improved performance may be obtained by carefully designing loss the function, as Shen's paper demonstrates [82]. We also want to explore the robustness of MDBoost. Since MDBoost considers the whole margin distribution, it is supposed to be more robust to outliers. More experiments are required to test this issue.

For the general framework of AnyBoost$_{\mathrm{TC}}$, we want to employ the framework for face recognition. In this way, we could use and compare various loss functions as well as the regularization terms. Higher accuracy is likely to be achieved when a appropriate optimization object is formed.

For the linear representation ensembles for face recognition, we want to apply the BLRC methods to extract some off-the-shelf features (*e.g.* HOG, SHIFT) instead of raw pixels for further improvement on accuracy. We also plan to use other ensemble learning methods such as random forests to combine the linear regression classifiers. Another possible direction is to introduce the hinge loss and the kernel trick from SVM to the linear representation ensemble. This way, the linear representation is actually performed on a higher-dimensional feature space where one usually can achieve higher discriminative power easily.

Bibliography

[1] P. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Trans. Inf. Theory*, 44(2):525–536, 1998.

[2] P. Bartlett, M. Jordan, and J. McAuliffe. Convexity, classification, and risk bounds. *J. Amer. Stat. Assoc.*, 101(473):138–156, 2004.

[3] J. Bi, T. Zhang, and K. P. Bennett. Column-generation boosting methods for mixture of kernels. In *Proc. ACM Int. Conf. Knowledge Discovery & Data Mining*, pages 521–526, Seattle, WA, USA, 2004. ACM.

[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[5] L. Breiman. Bagging predictors. *Mach. Leran.*, 24(2):123–140, 1996.

[6] L. Breiman. Arcing classifier. *Ann. Statist.*, 26(3):801–849, 1998.

[7] L. Breiman. Combining predictors. Technical report, Statistics Department, University of California, Berkeley, 1998.

[8] L. Breiman. Prediction games and arcing algorithms. *Neural Comp.*, 11(7):1493–1517, 1999.

[9] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001.

[10] E. J. Candès and M. B. Wakin. An introduction to compressive sampling. *IEEE Signal Process. Mag.*, 25(2):21–30, 2008.

[11] C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines, 2001. `http://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/`.

[12] N. Chawla and K. Bowyer. Random subspaces and subsampling for 2-d face recognition. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, volume 2, pages 582–589, 2005.

[13] J. Chien and C. Wu. Discriminant waveletfaces and nearest feature classifiers for face recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(12):1644–1649, 2002.

[14] M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, AdaBoost and Bregman distances. *Mach. Learn.*, 48(1-3):253–285, 2002.

[15] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operation Res.*, 8(1):101–111, 1960.

[16] A. Demiriz, K. Bennett, and J. Shawe-Taylor. Linear programming boosting via column generation. *Mach. Learn.*, 46(1-3):225–254, 2002.

[17] J. Demsar. Statisitcal comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, December 2006.

[18] J. Demsar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.

[19] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Comp.*, 10(7):1895–1923, 1998.

[20] C. Domingo and O. Watanabe. MadaBoost: A modification of AdaBoost. In *Proc. Annual Conf. Learn. Theory*, pages 180–189. Morgan Kaufmann, 2000.

[21] J. Duchi and Y. Singer. Boosting with structural sparsity. In *Proc. Int. Conf. Mach. Learn.*, pages 297–304, Montreal, Quebec, Canada, 2009. ACM.

[22] T. Evgeniou, M. Pontil, and A. Elisseeff. Leave one out error, stability, and generalization of voting combinations of classifiers. *Mach. Learn.*, 55(1):71–97, 2004.

[23] Y. Freund. An adaptive version of the boost by majority algorithm. *Mach. Learn.*, 43(3):293–318, 2001.

[24] Y. Freund and R. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Comput. learn. theory*, pages 23–37, 1995.

[25] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Mach. Learn.: Proceed. Thirteenth Inter. Conf.*, pages 148–156. MORGAN KAUFMANN PUBLISHERS, INC., 1996.

[26] Y. Freund and R. Schapire. Game theory, on-line prediction and boosting. In *Ninth Ann. Conf. on Comput. Learn. Theo.*, pages 325–332. ACM, 1996.

[27] Y. Freund, R. Schapire, Y. Singer, and M. Warmuth. Using and combining predictors that specialize. In *Proc. ACM Symp. Theory of Comp.*, pages 334–343, 1997.

[28] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comp. & Syst. Sci.*, 55(1):119–139, 1997.

[29] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Ann. Statist.*, 28(2):337–407, 2000.

[30] J. Friedman, T. Hastie, and R. Tibshirani. Rejoiner for additive logistic regression: A statistical view of boosting. *Ann. Statist.*, 28:400–407, 2000.

[31] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. Springer Series in Statistics, 2009.

[32] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 2001.

[33] A. Garg and D. Roth. Margin distribution and learning. In *Proc. Int. Conf. Mach. Learn.*, pages 210–217, Washington, DC, 2003.

[34] A. Georghiades, P. Belhumeur, and D. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(6):643–660, 2002.

[35] A. J. Grove and D. Schuurmans. Boosting in the limit: maximizing the margin of learned ensembles. In *Proc. National Conf. Artificial Intell.*, pages 692–699, Madison, Wisconsin, USA, 1998.

[36] V. Guruswami and A. Sahai. Multiclass learning, boosting, and error-correcting codes. In *Proc. Annual Conf. Learn. Theory*, pages 145–155. ACM, 1999.

[37] Z. Hao, C. Shen, N. Barnes, and B. Wang. Totally-corrective multi-class boosting. *Computer Vision–ACCV 2010*, pages 269–280, 2011.

[38] T. Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8):832–844, 2002.

[39] ILOG, Inc. CPLEX 11.1, 2008. `http://www.ilog.com/products/cplex/`.

[40] O. Kallenberg. *Foundations of Modern Probability*. Springer-Verlag, 1997.

[41] J. Kivinen and M. K. Warmuth. Boosting as entropy projection. In *Proc. Annual Conf. Learn. Theory*, pages 134–144, Santa Cruz, California, US, 1999. ACM.

[42] G. Lebanon and J. Lafferty. Boosting and maximum likelihood for exponential models. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 447–454. MIT Press, 2001.

[43] J. Leskovec. Linear programming boosting for uneven datasets. In *Proc. Int. Conf. Mach. Learn.*, pages 456–463, 2003.

[44] S. Z. Li and J. Lu. Face recognition using the nearest feature line method. *IEEE Trans. Neural Netw.*, 10:439–443, 1999.

[45] S. Z. Li and Z. Zhang. FloatBoost learning and statistical face detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1112–1123, 2004.

[46] C. Liu and H. Wechsler. Gabor feature based classification using the enhanced fisher linear discriminant model for face recognition. *IEEE Trans. Image Process.*, 11(4):467–476, 2002.

[47] H. Lodhi, G. J. Karakoulas, and J. Shawe-Taylor. Boosting the margin distribution. In *Proc. Int. Conf. Intelli. Data Eng. & Automated Learn., Data Mining, Financial Eng., & Intelli. Agents*, pages 54–59, London, UK, 2000. Springer-Verlag.

[48] J. Lu, K. Plataniotis, A. Venetsanopoulos, and S. Li. Ensemble-based discriminant learning with boosting for face recognition. *IEEE Trans. Neural Netw.*, 17(1):166–178, 2006.

[49] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operation Res.*, 53(6):1007–1023, 2005.

[50] A. Makhorin. Glpk (gnu linear programming kit), 2006.

[51] A. Martinez and R. Benavente. The AR face database. Technical report, CVC Technical report, 1998.

[52] L. Mason, J. Baxter, P. Bartlett, and M. Frean. *Functional gradient techniques for combining hypotheses*, chapter 12, pages 221–247. Advances in Large Margin Classifiers. MIT press, 1999.

[53] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 512–518, 2000.

[54] D. Mease and A. Wyner. Evidence contrary to the statistical view of boosting. *J. Mach. Learn. Res.*, 9:131–156, 2008.

[55] R. Meir and G. Rätsch. *An introduction to boosting and leveraging*, pages 118–183. Advanced lectures on machine learning. Springer-Verlag, New York, NY, USA, 2003.

[56] MOSEK ApS. The MOSEK optimization toolbox for matlab manual, version 5.0, revision 93, 2008. `http://www.mosek.com/`.

[57] I. Naseem, R. Togneri, and M. Bennamoun. Linear regression for face recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(11):2106–12, 2010.

[58] A. Y. Ng. Feature selection, ℓ_1 vs. ℓ_2 regularization, and rotational invariance. In *Proc. Int. Conf. Mach. Learn.*, pages 78–83, Banff, Alberta, Canada, 2004.

[59] P. Phillips, H. Wechsler, J. Huang, and P. Rauss. The FERET database and evaluation procedure for face-recognition algorithms. *Image Vis. Comp.*, 16(5):295–306, 1998.

[60] J. Quinlan. Bagging, boosting, and C4. 5. In *Proc. National Conf. Artificial Intell.*, pages 725–730, 1996.

[61] G. Rätsch, S. Mika, B. Schölkopf, and K.-R. Müller. Constructing boosting algorithms from SVMs: An application to one-class classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(9):1184–1199, 2002.

[62] G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. *Mach. Learn.*, 42(3):287–320, 2001. data sets are available at `http://theoval.cmp.uea.ac.uk/~gcc/matlab/index.shtml`.

[63] G. Rätsch, M. Warmuth, S. Mika, T. Onoda, S. Lemm, and K. Müller. Barrier boosting. In *Proc. Annual Conf. Learn. Theory*, pages 170–179, 2000.

[64] G. Rätsch and M. K. Warmuth. Efficient margin maximizing with boosting. *J. Mach. Learn. Res.*, 6:2131–2152, 2005.

[65] L. Reyzin and R. E. Schapire. How boosting the margin can also boost classifier complexity. In *Proc. Int. Conf. Mach. Learn.*, Pittsburgh, Pennsylvania, USA, 2006.

[66] R. M. Rifkin and R. A. Lippert. Value regularization and Fenchel duality. *J. Mach. Learn. Res.*, 8:441–479, 2007.

[67] R. T. Rockafella. *Convex Analysis*. Princeton University Press, 1997.

[68] S. Rosset, J. Zhu, and T. Hastie. Boosting as a regularized path to a maximum margin classifier. *J. Mach. Learn. Res.*, 5:941–973, 2004.

[69] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323, 2000.

[70] C. Rudin, I. Daubechies, and R. E. Schapire. The dynamics of AdaBoost: Cyclic behavior and convergence of margins. *J. Mach. Learn. Res.*, 5:1557–1595, 2004.

[71] C. Rudin, R. E. Schapire, and I. Daubechies. Analysis of boosting algorithms using the smooth margin function. *Ann. Statist.*, 35(6):2723–2768, 2007.

[72] R. Schapire. Using output codes to boost multiclass learning problems. In *Mach. Learn.*, pages 313–321. Citeseer, 1997.

[73] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Mach. learn.*, 37(3):297–336, 1999.

[74] R. E. Schapire. Theoretical views of boosting and applications. In *Proc. Int. Conf. Algorithmic Learn. Theory*, pages 13–25, London, UK, 1999. Springer-Verlag.

[75] R. E. Schapire. *The boosting approach to machine learning: An overview*, pages 149–172. Nonlinear Estimation and Classification. Springer, 2003.

[76] R. E. Schapire and Y. Freund. Improved boosting algorithms using confidence-rated predictions. *Mach. Learn.*, 37(3):297–336, Dec. 1999.

[77] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Ann. Statist.*, 26(5):1651–1686, 1998.

[78] S. Shalev-Shwartz and Y. Singer. On the equivalence of weak learnability and linear separability: New relaxations and efficient boosting algorithms. In *Proc. Annual Conf. Learn. Theory*, Helsinki, Finland, 2008.

[79] J. Shawe-Taylor and N. Cristianini. Further results on the margin distribution. In *Proc. Annual Conf. Learn. Theory*, pages 278–285, Santa Cruz, California, 1999.

[80] C. Shen and H. Li. On the dual formulation of boosting algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32:2216–2231, 2010.

[81] C. Shen and H. Li. On the dual formulation of boosting algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, pages 2216–2231, 2010.

[82] C. Shen, P. Wang, and H. Li. Lacboost and fisherboost: optimally building cascade classifiers. In *Proc. Eur. Conf. Comp. Vis.*, volume 2, pages 608–621, 2010.

[83] J. Sochman and J. Malas. AdaBoost with totally corrective updates for fast face detection. In *Proc. IEEE Int. Conf. Automatic Face & Gesture Recogn.*, pages 445–450, Seoul, Korea, 2004.

[84] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *J. Mach. Learn. Res.*, 7:1531–1565, 2006.

[85] R. Tibshirani. Regression shrinkage and selection via the Lasso. *J. Royal Stat. Soc. Ser. B (Methodological)*, 58(1):267–288, 1996.

[86] C. Tsallis. Possible generalization of Boltzmann-Gibbs statistics. *J. Stat. Physics*, 52:479–487, 1988.

[87] M. Turk and A. Pentland. Face recognition using eigenfaces. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 586–591, 1991.

[88] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[89] V. Vapnik. *The nature of statistical learning theory*. Statistics for Engineering and Information Science. Springer Verlag, Berlin, 2000.

[90] P. Viola and M. Jones. Robust real-time face detection. *Int. J. Comp. Vis.*, 57(2):137–154, 2004.

[91] P. Viola and M. J. Jones. Robust real-time face detection. *Int. J. Comp. Vis.*, 57(2):137–154, 2004.

[92] L. Wang, M. Sugiyama, C. Yang, Z. Zhou, and J. Feng. On the margin explanation of boosting algorithms. In *Proc. Annual Conf. Learn. Theory*, pages 479–490, 2008.

[93] X. Wang and X. Tang. A unified framework for subspace face recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1222–1228, 2004.

[94] X. Wang and X. Tang. Random sampling LDA for face recognition. *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2004.

[95] X. Wang and X. Tang. Random sampling for subspace face recognition. *Int. J. Comp. Vis.*, 70(1):91–104, 2006.

[96] M. K. Warmuth, K. Glocer, and S. V. Vishwanathan. Entropy regularized lp-boost. In *Proc. Int. Conf. Algorithmic Learn. Theory*, pages 256–271, Budapest, Hungary, 2008.

[97] M. K. Warmuth, J. Liao, and G. Rätsch. Totally corrective boosting algorithms that maximize the margin. In *Proc. Int. Conf. Mach. Learn.*, pages 1001–1008, Pittsburgh, Pennsylvania, 2006.

[98] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31:210–227, 2009.

[99] J. Wu, M. D. Mullin, and J. M. Rehg. Linear asymmetric classifier for cascade detectors. In *Proc. Int. Conf. Mach. Learn.*, pages 988–995, Bonn, Germany, 2005.

[100] Y. Xi, Z. Xiang, P. J. Ramadge, and R. Schapire. Speed and sparsity of regularized boosting. In *Proc. Int. Workshop Artificial Intell. & Statistics*, pages 615–622, Florida, US, 2009.

[101] R. Xiao and X. Tang. Joint boosting feature selection for robust face recognition. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, volume 2, pages 1415–1422, 2006.

[102] T. Zhang. Adaptive forward-backward greedy algorithm for sparse learning with linear models. In *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, Canada, 2008.

[103] T. Zhang and B. Yu. Boosting with early stopping: Convergence and consistency. *Ann. Statist.*, 33:1538–1579, 2005.

[104] C. Zhu, R. H. Byrd, and J. Nocedal. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, 1997.

[105] J. Zhu, S. Rosset, H. Zou, and T. Hastie. Multi-class adaboost. *Ann. Arbor.*, 1001:48109, 2006.

[106] H. Zou and M. Yuan. The f_∞-norm support vector machine. *Statistica Sinica*, 18:379–398, 2008.